UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Enginyeria Telemàtica

Hes·so VALAIS WALLIS

Contributions to Interoperability, Scalability and Formalization of Personal Health Systems

PhD Thesis by

Albert Brugués de la Torre

Submitted to the

UNIVERSITAT POLITÈCNICA DE CATALUNYA

To obtain the degree of

DOCTOR OF PHILOSOPHY

Co-advised by

Dr. Michael Ignaz Schumacher
University of Applied Sciences
Western Switzerland (HES-SO)

Dr. Josep Pegueroles Vallés
Universitat Politècnica de Catalunya
BarcelonaTech (UPC)

PhD Program in Telematics Engineering
Barcelona, November 2015

A dedication ...

Abstract

The ageing of the world's population combined with unhealthy lifestyles are contributing to a major prevalence of chronic diseases. This scenario poses the challenge of providing good healthcare services to that people affected by chronic illnesses, but without increasing its costs. A prominent way to face this challenge is through pervasive healthcare.

Research in pervasive healthcare tries to shift the current centralized healthcare delivery model focused on the doctors, to a more distributed model focused on the patients. In this context Personal Health Systems (PHSs) consists on approaching sampling technologies into the hands of the patients, without disturbing its activities of the daily life, to monitor patient's physiological parameters and providing feedback on their state. The use of PHSs involves the patients in the management of their illness and in their own well being too.

The development of PHSs has to face technological issues in order to be accepted by our society. Within them it is important to ensure interoperability between different systems in order to make them work together. Scalability it is also a concern, as their performance must not decrease when increasing the number of users. Another issue is how to formalize the medical knowledge for each patient, as different patients may have different target goals. Security and privacy are a must feature because of the sensitive nature of medical data. Other issues involve the the integration with legacy systems, and the usability of graphical user interfaces in order to encourage old people with the use these technologies.

The aim of this PhD thesis is to contribute into the state-of-the-art of PHSs by tackling together different of the above-mentioned challenges. First, to achieve interoperability we use the CDA standard as a format to encode and exchange health data and alerts related with the status of the patient. We show how these documents can be generated automatically through the use of XML templates. Second, we address the scalability by distributing the computations needed to monitor the patients over their devices, rather than performing them in a centralized server. In this context we develop the MAGPIE agent platform, which runs on Android devices, as a framework able to provide intelligence to PHSs, and generate alerts that can be of interest for the patients and the medical doctors. Third, we focus on the formalization of PHSs by providing a tool for the practitioners where they can define, in a graphical way, monitoring rules related with chronic diseases that are integrated with the MAGPIE agent platform. The thesis also explores different ways to share the data collected with PHSs in order to improve the outcomes obtained with the use of this technology. Data is shared between individuals following a Distributed Event-Based System (DEBS) approach, where different people can subscribe to the alerts produced by the patient. Data is also shared between institutions with a network protocol called MOSAIC, and we focus on the security aspects of this protocol.

The research in this PhD focuses in the use case of Diabetes Mellitus; and it has been developed in the context of the projects MONDAINE, MAGPIE, COMMODITY$_{12}$ and TAMESIS.

Resumen

El envejecimiento de la población mundial combinado con unos estilos de vida no saludables contribuyen a una mayor prevalencia de enfermedades crónicas. Este escenario presenta el reto de proporcionar unos buenos servicios sanitarios a las personas afectadas por estas enfermedades, sin incrementar sus costes. Una solución prometedora a este reto es mediante la aplicación de lo que en inglés se denomina *pervasive healthcare*.

La investigación en este campo trata de cambiar el actual modelo centralizado de servicios sanitarios enfocado hacia el personal sanitario, por un modelo distribuido enfocado hacia los pacientes. En este contexto, los *Personal Health Systems* (PHSs) consisten en poner al alcance de los pacientes las tecnologías de monitorización, y proporcionarles información sobre su estado. El uso de PHSs involucra a los pacientes en la gestión de su enfermedad y en su propio bienestar.

La aceptación de los PHSs por parte de la sociedad implica ciertos retos tecnológicos en su desarrollo. Es importante garantizar su interoperabilidad para que puedan trabajar conjuntamente. Su escalabilidad también se debe tener en cuenta, ya que su rendimiento no tiene que verse afectado al incrementar su número de usuarios. Otro aspecto a considerar es cómo formalizar el conocimiento médico para cada paciente, ya que cada uno puede tener objetivos distintos. La seguridad y privacidad son características deseadas debido a la naturaleza sensible de los datos médicos. Otras problemáticas implican la integración con sistemas heredados, y la usabilidad de las interfaces gráficas para fomentar su uso entre las personas mayores.

El objetivo de esta tesis es contribuir al estado del arte de los PHSs tratando de manera conjunta varios de los retos mencionados. Para abordar la interoperabilidad se usa el estándar CDA como formato para codificar los datos médicos y alertas relacionados con el paciente. Además se muestra como estros documentos pueden generarse de forma automática mediante el uso de plantillas XML. Para tratar la escalabilidad se distribuye la computación para monitorizar a los pacientes en sus terminales móbiles, en lugar de realizarla en un servidor central. En este contexto se desarrolla la plataforma de agentes MAGPIE como *framework* para proporcionar inteligencia a los PHSs y generar alertas de interés para el médico y el paciente. La formalización se aborda mediante una herramienta que permite a los médicos definir de manera gráfica reglas de monitorización relacionadas con enfermedades crónicas, que ademas están integradas con la plataforma de agentes MAGPIE. La tesis también explora distintas formas de compartir los datos recolectados con un PHS, con el fin de mejorar los resultados obtenidos mediante esta tecnología. Los datos se comparten entre individuos siguiendo un enfoque de sistemas distribuidos basados en eventos (DEBS), donde distintos usuarios pueden suscribirse a las alertas producidas por el paciente. Los datos también se comparten entre instituciones mediante un protocolo de red llamado MOSAIC. En la tesis se desarrollan los aspectos de seguridad de este protocolo.

La tesis se centra en la Diabetes Mellitus como caso de uso, y se ha realizado en el contexto de los proyectos MONDAINE, MAGPIE, COMMODITY$_{12}$ y TAMESIS.

Resum

L'envelliment de la població mundial combinat amb uns estils de vida no saludables contribueixen a una major prevalença d'enfermetats cròniques. Aquest escenari presenta el repte de proporcionar uns bons serveis sanitaris a les persones afectades per aquestes enfermetats, sense incrementar-ne els costos. Una solució prometedora a aquest repte és mitjançant l'aplicació del que en anglès s'anomena *pervasive healthcare*.

L'investigació en aquesta camp tracta de canviar l'actual model centralitzat de serveis sanitaris enfocat en el personal sanitari, per un model de serveis distribuït enfocat en els pacients. En aquest context, els *Personal Health Systems* (PHSs) consisteixen en posar a l'abast dels pacients les tecnologies de monitorització, i proporcionar-los informació sobre el seu estat. L'ús de PHSs involucra els pacients en la gestió de la seva enfermetat i del seu propi benestar.

L'acceptació dels PHSs per part de la societat implica certs reptes tecnològics en el seu desenvolupament. És important garantir la seva interoperabilitat per tal de que puguin treballar conjuntament. La seva escalabilitat també s'ha de tenir en compte, ja que el seu rendiment no s'ha de veure afectat al incrementar-ne el número d'usuaris. Un altre aspecte a considerar és com formalitzar el coneixement mèdic per cada pacient, ja que cada un d'ells pot tenir objectius diferents. La seguretat i privacitat són característiques desitjades degut a la naturalesa sensible de les dades mèdiques. Altres problemàtiques impliquen la integració amb sistemes heretats, i la usabilitat de les interfícies gràfiques per fomentar-ne el seu ús entre les persones grans.

L'objectiu d'aquesta tesi és contribuir a l'estat de l'art dels PHSs tractant de manera conjunta varis dels reptes mencionats. Per abordar l'interoperabilitat s'utilitza l'estàndard CDA com a format per codificar les dades mèdiques i alertes relacionades amb el pacient. A més es mostra com aquests documents poden generar-se de forma automàtica mitjançant l'ús de plantilles XML. Per tractar l'escalabilitat es distribueixen les computacions per monitoritzar els pacients entre els seus terminals mòbils, en comptes de realitzar-les en un servidor central. En aquest context es desenvolupa la plataforma d'agents MAGPIE com a *framework* per proporcionar intelligència als PHSs i generar alertes d'interès per al metge i el pacient. La formalització s'aborda mitjançant una eina que permet als metges definir de manera gràfica regles de monitorització relacionades amb enfermetats cròniques, que a més estan integrades amb la plataforma d'agents MAGPIE. La tesi també explora diferents maneres de compartir les dades recol·lectades amb un PHS, amb l'objectiu de millorar els resultats obtinguts amb aquesta tecnologia. Les dades es comparteixen entre individus seguint un enfoc de sistemes distribuïts basats en events (DEBS), on diferents usuaris poden subscriure's a les alertes produïdes per el pacient. Les dades també es comparteixen entre institucions mitjançant un protocol de xarxa anomenat MOSAIC. A la tesi es desenvolupen els aspectes de seguretat d'aquest protocol.

La test es centra en la Diabetis Mellitus com a cas d'ús, i s'ha realitzat en el context dels projectes MONDAINE, MAGPIE, COMMODITY$_{12}$ i TAMESIS.

Acknowledgements

Contents

List of Figures

List of Tables

List of Acronyms

AAL	Ambient Assisted Living
ADE	Adverse Drug Events
AI	Artificial Intelligence
APIM	Agent Platform Independent Model
ATC	Anatomical Therapeutic Chemical classification system
BAN	Body Area Network
BDI	Belief-Desire-Intention
CCDB	Central Clinical DataBase
CDA	Clinical Document Architecture
CDA R2	Clinical Document Architecture Release 2
CGM	Continuous Glucose Monitoring
DEBS	Distributed Event-Based System
DM	Diabetes Mellitus
EC	Event Calculus
ECG	Electrocardiogram
EHR	Electronic Health Record
EIVL	Event-related periodic InterVaL of time
EU	European Union
GCM	Google Cloud Messaging
GDM	Gestational Diabetes Mellitus
GTS	General Timing Specification
HIS	Hospital Information System
HL7	Health Level 7
ICD	International Classification of Diseases

IEEE	Institute of Electrical and Electronics Engineers
IHE	Integrating the Healthcare Enterprise
JADE	Java Agent DEvelopment Framework
JSON	JavaScript Object Notation
LOINC	Logical Observation Identifiers Names and Codes
MAS	Multi-Agent Systems
MBU	Mobile Base Unit
MCC	MultiCast Contributor
MCP	MultiCast Petitioner
MIB	Medical Information Bus
MOA	Micro-agents On Android
OWL	Web Ontology Language
P2P	Peer-to-peer
PDA	Personal Digital Assistance
PHS	Personal Health System
RFID	Radio Frequency IDentification
RIM	Reference Information Model
SNOMED CT	Systemized NOmenclature of MEDicine Clinical Terms
UCC	UniCast Contributor
UCP	UniCast Petitioner
UI	User Interface
XML	eXtensible Markup Language
YP	Yellow Pages

Chapter 1

Introduction

"A journey of a thousand miles begins with a single step."
Laozi.

Contents

Pervasive healthcare is an emerging scientific discipline that involves the use of the ubiquitous computing technology (pervasive computing) in the healthcare environment [7]. As defined by Varshney, pervasive healthcare is the "*healthcare to anyone, anytime, and anywhere by removing locational, time and other restrains while increasing both the coverage and the quality of healthcare*" [137].

The health challenges of the OECD countries [106] are a motivation for doing research on the pervasive healthcare discipline. These challenges can be summarized as the following:

- There is a huge increase of the ratio between elderly and young people.

- Healthcare costs and the chronic diseases are increasing as people grew older.
- Current lifestyles (e.g. smoking, obesity, and inactivity) are contributing to the prevalence of chronic diseases.
- The constantly expansion of the scope of medicine contributes to increase both, the healthcare costs and the average life expectancy.
- There is an increasing lack of clinical professionals due to retirement and small number of medical and nursing students.

Moreover, research on pervasive healthcare is also motivated by the Europe 2020 program, a 10-year strategy proposed by the European Commission on 2010 for advancement of the economy of the European Union (EU). An integral part of Europe 2020 is to promote good health [54], and to do so four among the seven flagship initiatives of Europe 2020 are relevant to the healthcare domain, which are the following:

- **Innovation Union:** aims to maximize EU's capacity for innovation. In the healthcare domain the goal is to make Europe a world-leader in developing ways to promote active and healthy aging.
- **Digital Agenda for Europe:** is focused on developing and using digital applications. There are four key actions related with health:

  1. Give Europeans secure online access to their medical health data and achieve widespread telemedicine deployment.
  2. Propose a recommendation to define a minimum common set of data.
  3. Foster EU-wide standards, interoperability testing and certification of eHealth.
  4. Reinforce the AAL Joint Programme.

- **Agenda for new skills and jobs:** focuses on highlight the economic role of mental health and the health of the workforce. This should improve working conditions and workplaces that prioritize the health and well-being of their employees.
- **European platform against poverty:** aims to ensure economic and social cohesion. The European Commission will contribute by boosting efforts on health promotion and prevention with a focus in reducing health inequality.

The pervasive healthcare discipline tries to find new solutions and approaches to mitigate these challenges. To achieve this goal pervasive healthcare tries to modify the healthcare service delivery model in the Western countries, by moving it from a centralized approach focused on doctors to a decentralized one based on the patients [7]. In other words, it tries to move from a

reactive model in which people go to hospital because they are ill, to a pro-active and preventive model where people are active participants in their own well-being, and thus providing a more personalized healthcare. This personalization involves the use of technologies that can move the patient treatment and care from hospitalization to home. It also involves a continuous monitoring of vital signs (e.g. blood sampling, blood pressure, etc.) done automatically by the patient, rather than the periodic sample done inside the hospitals.

1.1 Application Environments of Pervasive Healthcare

Research in pervasive healthcare is focused on different topics that can be classified by their environment of application. Hence we have these four main environments: i) *Personal Health Systems* (PHSs) focused on monitoring vital signs and activities of the patient; ii) *Ambient Assisted Living* (AAL) *technologies* to help people live more independent; iii) *pervasive computing for hospitals* help the medical staff to coordinate their work; and iv) *persuasive technologies* to encourage people to have a healthier live.

1.1.1 Personal Health Systems (PHSs)

The focus of PHSs is to move the sampling technologies from the labs to the hands of the patient. To do so, one of the main goals is to design and develop reliable and non-intrusive wearable sensors that do not disturb the daily activities of the patient, and as well sensors with more powerful communication and processing mechanisms.

As shown in Figure 1.1 typical applications consist of a common architecture with three different tiers namely: the Body Area Network (BAN) (Tier 1), a gateway (Tier 2), and a remote monitoring system (Tier 3). The remote monitoring system is usually a server hosted at the hospital side which collects, stores and eventually analyzes health-related information. The gateway can be a smartphone or a tablet, which acts as a relay node between the BAN and the remote monitoring system, while the BAN consists of a set of sensors deployed or implanted on the body of the patient. The goal of the BAN can be:

- The continuous monitoring of vital signs [143] such as the electrocardiogram (ECG), the heart rate, the oxygen saturation (SpO_2), the blood pressure, among others to detect anomalous situations.
- The detection of the activities of daily life such as walking, standing, sitting, reading, eating or lying, when combining sensor networks with machine learning approaches [12].

On both cases the aim is to produce safety, assistance and early warnings to the patient. This help can be extended by informing both the patient's emergency contacts [92] and the medical doctor in charge of the patient [28] when a dangerous situation is detected. PHSs can also be used to mitigate diseases on an early stage in order to prevent them from occurring [108].

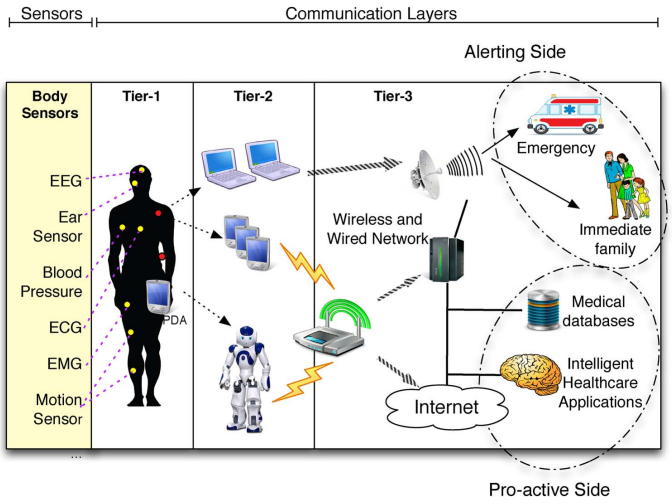


Figure 1.1: Three tier architecture of a PHS (from [3] ©2013 IEEE)

The development of PHSs have specific challenges such as the transmission of vital signs over limited and variable capacity wireless networks. The monitoring system should operate autonomously, and it must be taken into account that when the number of patients using the system grows it can increase significantly the amount of network traffic depending on the number of sensed vital signs, and their upload frequency. This increase of traffic can affect the reliability of message delivery to the remote monitoring system [137].

Examples of PHSs include among others, a fall detection system based on a smartphone which uses an accelerometer worn on the waist [1]; a rehabilitation support system for the elderly using a set of sensors to track the movement, the ECG and the breath rate of the patient [110]; the detection of early signs of dementia based on the eye tracking performed with electrodes [139]; novel applications such as automatic dietary monitoring which can be done using different techniques based on sensors [6]; and support applications based on agents to provide assistance in form of diagnostic advice [99].

1.1.2 Ambient Assisted Living (AAL) Technologies

The goal of AAL technologies is to support the daily lives of people in general (e.g. elderly, ill, disabled), to help them to have an independent life with respect to wellness, health and safety. In contrast with PHSs in AAL technologies sensors are placed on the environment rather than on the body of the patient. This approach gives a contextual information that can be used to complement the information provided by the BAN of a PHS [52].

When some kind of intelligence is added to the sensing environment, this is known as "Ambient Intelligence" [45]. In these intelligent environments the tasks are not just to sense the current conditions of the patient, but analyzing them and deciding if a reaction is needed and which one [17]. A subject of current research, in both PHS and AAL technologies, is to de-

Figure 1.2: Example of an AAL in a laboratory environment (from [87] with permission of Springer Science+Business Media)

termine how this intelligence is accomplished, where it is provided and how it is managed. To address the scalability of the system the raw data can be analyzed by the sensors themselves, rather than send it every time to a central server for further processing. This could lead to a shorter life time of the battery that can be compensated by the reduction in the amount of communications needed. On the other hand, it must be taken into account that in some applications the processing capabilities of the sensors might be not enough.

Some research efforts in AAL technologies are focused on the so called "smart homes", where the sensing environment is the house of the patient. The smart homes are equipped with different sensing, interaction, and assistance facilities to provide the technology and measurement environment for an AAL scenario. As in the example shown in Figure 1.2, these can include: ambient sensors integrated in switches, blinds or power sockets; position tracking solutions (e.g. smart carpet equiped with Radio Frequency IDentification (RFID)); intelligent household appliances (e.g. refrigerator that warn about spoiled food); an autonomous robot transportation platform; among others. All these technologies can be used to proactively assist people through early detection of problems, however the use of these assistive devices transfer the dependence from human side to machinery side which reduces social connections of assisted people [127].

Assisted living systems promise a huge potential for handicapped and elderly people, but they must meet these three requirements [87]:

1. They have to be ambient and unobtrusive to reach high acceptance.

2. They have to adapt themselves to changing personal situations or capabilities of the indi-

vidual and the environment to fulfill individual needs.

3. They have to provide their services in an accessible way to enhance usability.

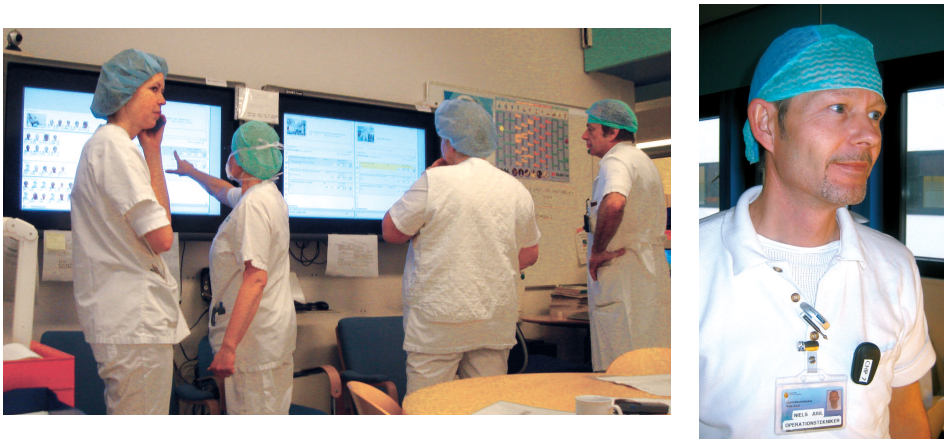
Examples of AAL implementations include among others, in-home monitoring of healthy independent elders [27]; monitoring falls in home of elderly [144]; the eCAALYX system, a solution to reduce morbidity and mortality of elderly people suffering from comordibity [113]; and several applications based on agents such as a home health system to identify chronic conditions from the patterns of symptoms [36]; a smart home in which specific scenarios such as insomnia are detected by means of scenario recognizer agents [115]; and the NOCTURNAL system in which patients are monitored while sleeping using ambient sensors and agents [34].

1.1.3 Pervasive Computing for Hospitals

The working environment of a hospital and the tasks performed on it are very different from the ones executed inside an office, therefore current computing technology fit very poorly on hospitals [13]. The goal of placing pervasive computing inside hospitals is to help clinicians to coordinate their work in an efficient manner. To do so, these technologies must provide easy and secure access to data, without cumbersome login procedures, and fast navigation in the large datasets, and taking into account that physicians must have access to many applications in different locations [14].

The new kinds of pervasive computing technologies suited for hospitals are mostly used for i) the advanced management of health information (e.g., patient health records, pharmaceutical information, etc.) and ii) provision of location and context-aware services [50]. This allow clinical personnel easy access to relevant clinical data in specific situations. For example, the use of location provision based on Bluetooth beacons in combination with mobile phone applications, can make easier the task of finding a surgeon that is not currently in surgery and inform him about the new surgery [65]. The RFID technology in combination with agent technology can also be used to assist on the planification of the nurses work, which in turn helps them to gain time [46].

Different projects have been carried out to introduce pervasive computing inside the hospitals. The Activity-Based Computing project [14] proposes a framework based on activities. An activity is a collection of tasks, supporting multiple users as participants, and can be suspended and resumed over time and space. The iHospital system [65], a hospital scheduling and awareness system which incorporates location tracking, large interactive displays, and mobile phones. The infrastructure is based on the application *AwareMedia* that shows information about the work in different operating rooms, and the *AwarePhone* a program that provides an overview of people at work and the status of surgeries in the operating rooms. Other example is [90], in which two embedded systems are developed, an ubiquitous echograph and a networked digital camera by replacing the bus of the devices with the network of the Hospital Information System (HIS).



(a) Interactive displays **(b)** Bluetooth beacon

Figure 1.3: Examples of pervasive technologies used in a hospital (from [65] ©2006 IEEE)

1.1.4 Persuasive Technologies

The idea of persuasive technologies is to use the technology to motivate desirable behaviors. B. J. Fogg was the originator of this field of research by coining the term "captology" [57], an acronym for "computers as persuasive technologies", which explores the overlap between persuasion in general and computing technology.

Persuasive technologies applied into the healthcare domain covers a gap left by PHSs and AAL technologies, as some of the diseases that people suffer are caused by inappropriate lifestyles such as smoking, drinking, inactivity or stress. In such cases the monitoring of the patients is not enough. The goal of persuasive technologies applied in the healthcare domain is to shift the healthcare paradigm from managing illness to maintaining wellness, by seeking to alter peoples' lifestyles [13]. These type of systems help users by motivating them towards the improvement of their health habits, and by enabling them to monitor and visualize their behaviors, reminding them to perform specific tasks, and recommending healthier behaviors or actions [15]. A strategy to achieve all these goals is through gamification, which consists on using game elements in non-game context [48].

A specific challenge for the designers of these technologies is the consideration of different users' personality types, rather than design applications for general audience, to sustain user interest over time [64].

Examples of already implemented persuasive systems include among others, mobile games designed to increase teenagers' physical activity [9]; a personal digital assistant based on relational agents that encourage people to walk more [20]; the MONARCA system, a persuasive system for bipolar patients based on Android [15]; and the application HealthyTogether that promotes physical activities through games [39].

1.2 Challenges in Pervasive Healthcare

Pervasive healthcare has two main general challenges that are i) to provide better healthcare services to an increasing number of people; and ii) to reduce the long-term cost of the healthcare services [136].

On the other hand, pervasive healthcare has specific technological challenges related with the following areas:

- **Interoperability:** most of the existing implementations do not interoperate well [17], resulting in segmented solutions that are highly specific in nature, often known as the so called "closed" systems. These systems are not able to communicate with other components in order to support a collaborative behavior. A suitable way to achieve interoperability is using healthcare standards such as Health Level 7 (HL7) for messages, and/or Systemized NOmenclature of MEDicine Clinical Terms (SNOMED CT) for ontology representation, among others.

- **Scalability:** in systems with an extremely large number of patients, regular physiological data might have to be stored and analyzed. Such systems should be horizontally scalable, that is, the performance of the system must scale linearly when adding new patients [103].

- **Formalization of domain knowledge:** in those implementations with clinical decision support the domain knowledge needs to be transformed for machine processing. In many cases, this knowledge can be difficult to formalize and depends mostly on the expertise of medical staff [87].

- **Privacy and security:** among other things healthcare data should be available anytime anywhere, but only to authorized persons [11]. The devices must provide an authentication interface for the legal users of the pervasive system [95]. The requirements on privacy and security are to avoid, for example, that insurance companies refuse coverage for people with poor health.

- **Integration with legacy systems:** new pervasive healthcare systems may coexist with other already implemented solutions like the HISs of an hospital. This new solutions should be compatible with the old ones [112].

- **Mobility:** in environments with a large amount of heterogeneous devices and a wide deployment scale, a crucial issue is to properly describe and discover available healthcare services taking into account the dynamic operational and environmental context interactions [132].

- **Usability:** pervasive healthcare systems should provide intuitive interfaces to encourage people their use. Those who are less familiar with technology are generally willing to use intelligent mobile devices if these devices allow them to live more independently [137].

- **Legal and regulatory implications:** legal frameworks already exist for protecting sensitive medical information, and therefore all the solutions are required to address these laws. Those legal frameworks are the "Health Insurance Portability and Accountability Act of 1996" (HIPAA) in the USA, and the "Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data" in Europe.

1.3 Thesis Scope and Contributions

This PhD thesis is focused on contributing to some of the previously mentioned challenges in the context of PHSs. In particular, the contributions deals with the interoperability, scalability and formalization of domain knowledge of these systems. The thesis also explores ways to extend the functionalities of PHSs at Tier 3 by introducing concepts from DEBS into PHSs, and also security aspects of a protocol to exchange information collected with these systems.

The work of this thesis is centered in the use case of Diabetes Mellitus (DM). DM is a chronic health condition in which the pancreas does not produce enough insulin, or the body cannot use it effectively. The former is known as DM Type I and patients belonging to this group need external administration of insulin, the latter is DM Type II and does not usually require external administration of insulin. A third type is Gestational DM, which can appear during the pregnancy. In all cases the complications may lead to have episodes with high blood glucose levels (hyperglycemia) or low blood glucose levels (hypoglycemia). A person with DM can develop long-term complications such as damage to small blood vessels in the kidneys (nephropathy) or in the eyes (retinopathy), damage to nerves (neuropathy), and twice the risk of having cardiovascular disease [31]. The prevention and treatment of DM includes a healthy diet, a regular physical activity, maintaining a normal body weight, and not smoking. It is estimated that DM will affect 439 million adults by 2030 [123], and that health expenditures for DM will be USD 490 billion by the same year [148].

In this thesis we introduce extensions to the architecture of common PHSs that we can find in the literature in the three above-mentioned dimensions. Figure 1.4 depicts the architecture of the PHS reported in the thesis. As shown, the system has two different actors: patients and medical doctors. The medical side of this system corresponds to the Tier 3 of a PHS. In this case, healthcare professionals can interact with the system by means of a web application that allows them to visualize and analyze data from their patients. In addition, they can **formalize the medical knowledge** by defining specific **monitoring rules** for each patient. These rules deal with temporal patterns of physiological parameters that are expressed as first order logic predicates. Regarding the patient side, it maps to the Tiers 1 and 2, where patients are monitored by means of an Android based smartphone (Tier 2) and a set of sensors that conforms the BAN (Tier 1). This monitoring task is carried out by means of the **MAGPIE agent platform** that has been developed as a part of this thesis. This agent platform allows the development of a Multi-

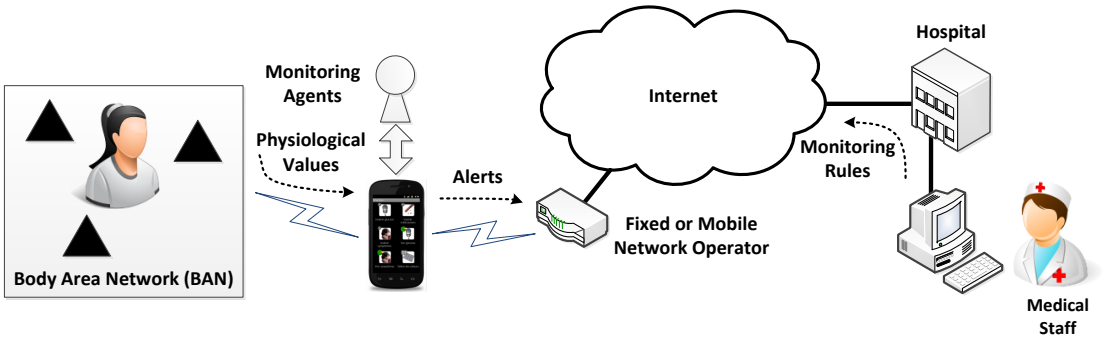


Figure 1.4: Architecture of the PHS developed in the realization of this PhD thesis

Agent Systems (MAS) running in Android, which is able to perceive the physiological values measured by the sensors of the BAN. The **monitoring agents** in **MAGPIE** are responsible to perform reasoning on these data and to produce **alerts** according to the **monitoring rules** defined by the medical doctors. Thus, the system becomes reconfigurable at runtime by the medical experts. This design strategy in the system's architecture, where the computations are done in Tier 2 rather than in Tier 3, is expected to improve the **scalability** of the system, as Tier 3 is a component shared by all the patients of and PHS and can consequently become a bottleneck. Finally, the traffic flow between Tiers 2 and 3 takes into account **interoperability** as it is done using standards like HL7 and JavaScript Object Notation (JSON). In particular, in the communications flowing from Tier 2 to Tier 3 the Clinical Document Architecture (CDA) standard is used to encapsulate the **alerts** along with the physiological data that produced them. For the traffic flowing from Tier 3 to Tier 2, the **monitoring rules** are represented and sent in JSON notation.

Furthermore, we explore ways of sharing and exchanging the healthcare information collected in our PHS. In particular we take into account two different ways of exchanging this information: between individuals and between institutions. In the former, we integrate the use of **MAGPIE** in a DEBS-like system to report events that are actually alerts related with the status of the patient. DEBSs implement the publish/subscriber pattern, which in our case means that the patient becomes a publisher of her data, and people like medical doctors, relatives, or another patients with the same disease can act as subscribers that consume these data. In the latter, we define a network protocol based on MAS that we call MOSAIC. The goal of this protocol is to exchange information between healthcare institutions by supporting multilateral agreements. In this case we specifically focus on the security aspects of this protocol.

As shown in Figure 1.5, different of the above-mentioned concepts are related to each other and represent the core contributions of the thesis, which are listed below:

- We show how CDA documents that capture the status of the patient can be generated automatically in the Tier 2 of a PHS. To achieve that, we use a strategy that consist in

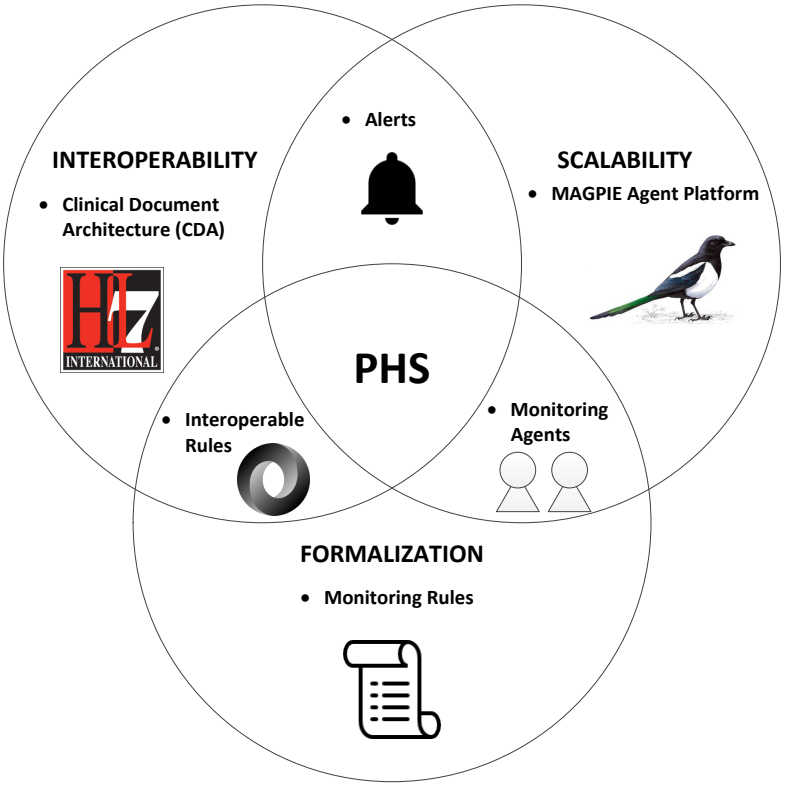


Figure 1.5: Contributions of this PhD thesis in the context of PHSs

using a set of eXtensible Markup Language (XML) templates that can be filled with the proper values and matched together.

- We use the JSON standard within the healthcare domain to express monitoring rules that are used to report alerts on the status of the patient.
- We develop the MAGPIE agent platform. In this agent platform we model concepts from MAS like agents and the agent environment, and we have designed it to run on Android devices. This platform is a reusable software solution to be used in Tier 2 of PHSs to continuously monitor patients affected by chronic diseases.
- We prove that the best practice towards scalable PHSs is to put the reasoning part close to the patient, by taking advantage of the computation capabilities of current smartphones and tablets available in the market.
- We provide a web application where medical doctors can formalize their medical knowledge to monitor their patients. This application is based on the definition of monitoring rules in a graphical way. The rules are based on the combination of different events to build temporal patterns of physiological parameters that can be run by the agents in MAGPIE.

- We show how with the monitoring rules that we can define graphically, we are able to detect temporal patterns in a real dataset of patients suffering DM Type II.

In addition to this, we complete our contributions in PHSs focusing on different usages that can be given to Tier 3, which are:

- We propose to use the Tier 3 of a PHS implemented with MAGPIE as an event notification service, which implies that the PHS implements the publish/subscriber communication paradigm. With this proposal, the patient becomes a publisher of her healthcare date, and that medical doctors, relatives, and other patients subscribers to the alerts produced by the patient.
- We propose a protocol called MOSAIC, which is based on MAS, to exchange information between nodes that can be healthcare institutions. This protocol supports multilateral agreements for the exchange of information and we analyze the security aspects of this protocol.

1.4 List of Publications

In this section we list the different publications related with the realization of this PhD thesis.

Articles in JCR/Scopus indexed journals

- A. Brugués, S. Bromuri, M. Barry, Ó. J. del Toro, M.R. Mazurkiewicz, P. Kardas, J. Pegueroles, and M. Schumacher. Processing diabetes mellitus composite events in MAGPIE. *Journal of Medical Systems*, (in press).
- M. Lluch-Ariet, A. Brugués, F. Vallverdú, and J. Pegueroles. Knowledge sharing in the health scenario. *Journal of Translational Medicine*, 12(Suppl 2): S8, 2014.

Book chapters

- A. Brugués, J. Pegueroles, S. Bromuri, and M. Schumacher. Current trends in interoperability, scalability and security of pervasive healthcare systems. In K. Curran, editor, *Recent Advances in Ambient Intelligence and Context-Aware Computing*, Advances in Computational Intelligence and Robotics (ACIR), chapter 14, pages 227-247. IGI Global, 2015.

International conference papers

- A. Brugués, S. Bromuri, J. Pegueroles, and M. Schumacher. Providing interoperability to a pervasive healthcare system through the HL7 CDA standard. In *Proc. 15th International HL7 Interoperability Conference (IHIC)*, pp. 5-12, 2015.

- A. Brugués, M. Lluch-Ariet, and J. Pegueroles. Security analysis of a protocol based on multiagents systems for clinical data exchange. In *Proc. 7th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, pp. 305-311, 2013.

Workshop papers

- A. Brugués, S. Bromuri, J. Pegueroles, and M. Schumacher. MAGPIE: An agent platform for the development of mobile applications for pervasive healthcare. In *Proc. 3rd International Workshop on Artificial Intelligence and Assistive Medicine (AI-AM/NetMed)*, pp. 6-10, 2014.

- M. Lluch-Ariet, A. Brugués, and J. Pegueroles. Performance evaluation of MOSAIC: A multi agent system for multilateral exchange agreements of clinical data. In *Proc. 7th International Workshop on Agents Applied in Health (A2HC)*, pp. 7-17, 2012.

Posters

- A. Brugués, S. Bromuri, J. Pegueroles, and M. Schumacher. Towards a personalizable health system for diabetes. Poster presented at: *2nd. International Conference on Biomedical and Health Informatics (BHI)*, 2014.

National conference papers

- A. Brugués, S. Bromuri, J. Pegueroles, and M. Schumacher. MAGPIE: Mobile computing with agents and publish/subscribe for intelligent u-healthcare. In *Swiss Medical Informatics Vol 31, 2015*.

- A. Brugués, M. Schumacher, J. Pegueroles, and S. Bromuri. Proporcionando interoperabilidad a un sistema ubicuo de asistencia médica mediante el estándar HL7 CDA. In *Proc. XI Jornadas de Ingeniería Telemática (JITEL)*, pp. 249-256, 2013.

- A. Brugués, M. Lluch-Ariet, and J. Pegueroles. Análisis de suguridad de un protocolo para el intercambio de datos clínicos basado en sistemas multiagente. In *Proc. XII Reunión Espanola sobre Criptografía y Seguridad de la Información (RECSI)*, pp. 217-224, 2012.

Project deliverables

- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MAGPIE D5: Tutorials and API Guides, 2015.

- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MAGPIE D4: MAGPIE integration with BioHarness and Distributed Event Based Systems, 2015.

- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MAGPIE D3: The MAGPIE Agent Platform, 2015.

- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MAGPIE D2: Analysis of Requirements for an Agent Platform for Personal Health Systems, 2015.
- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MAGPIE D1: Final Report, 2015.
- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MONDAINE (D4). Towards Individualized Complex and Sequential Event Processing for GDM Patients, 2014.
- A. Brugués, S. Bromuri, and M. Schumacher. Project Report MONDAINE (D2). The MONDAINE System and its Cognitive Models: Towards a Scalable and Iteroperable Pervasive Healthcare System, 2013.

1.5 Thesis Structure

This thesis is composed of seven chapters. In this first chapter we introduced the pervasive healthcare research field by identifying its different areas of application, and the challenges that must be faced when developing a pervasive healthcare systems. We have seen that in this thesis we tackle some of the identified challenges focusing them on the case of PHSs. More specifically, we described the contributions of this thesis and listed its related scientific publications. The rest of the chapters cover the following topics:

- Chapter 2 reviews the state of the art of PHSs in the three dimensions studied in this thesis: interoperability, scalability and formalization of medical knowledge. This chapter also reviews different existing agent platforms for Android, and identifies the differences between the literature and the research conducted in this thesis.
- Chapter 3 is about our proposal on how to deal with interoperability when developing PHSs. The chapter describes how CDA documents can be generated in the Tier 2 of a PHS for the use case of Gestational Diabetes Mellitus (GDM), and evaluates the proposed solution when considering two different strategies for generating these documents.
- Chapter 4 focuses on the scalability by describing the MAGPIE agent platform, and the concepts built around it like the agents, and the agent environment. This chapter also shows how the platform can work with sensors for measuring physiological values, and in particular with the Zephyr's BioHarness one. This chapter also evaluates the scalability in PHSs comparing the MAGPIE approach, where the computations needed by the agents for monitoring the patient are done in Tier 2, against an approach where the computations are done in Tier 3.
- Chapter 5 describes the proposal for modeling medical knowledge through temporal patterns in a graphical way. First, the chapter shows the different patterns that we model in

terms of the temporal domain, and how these patterns are related with its graphical representation. Second, the chapter shows how the previously introduced graphical patterns are mapped to the underlying logic used by the agents in MAGPIE. The chapter concludes by showing different temporal patterns for the use case of DM Type II, and evaluating the detection of these patterns retrospectively in a dataset of patients affected by this disease.

- Chapter 6 is about different usages for Tier 3 in order to share the health information produced in PHSs. This chapter is divided in two parts. The first part describes the use case of exchanging information between individuals, where the Tier 3 of a PHS developed with MAGPIE is used as an event notification service. In particular it shows the different scenarios that this service can handle. The second part describes the use case of exchanging information between institutions. In this part the MOSAIC protocol is introduced, and by analyzing the different threats to it, an architecture to make it secure is derived.
- Chapter 7 summarizes all the achievements of this thesis, and states possible lines of research to extend it.

Chapter 2

State of the Art

Contents

This chapter reviews the literature regarding pervasive healthcare systems, but focusing specifically on the three identified challenges that are subject of research in this thesis: interoperability (Section 2.1), scalability (Section 2.2) and formalization (Section 2.3). As stated in the previous chapter, we consider these three challenges in conjunction, which results in the development of the MAGPIE agent platform as a strategy for improving the scalability in Personal Health Systems (PHSs). Thus, other already existing agent platforms for Android are analyzed, in the scalability section, to identify their missings in relation with our work.

2.1 Interoperability in the Healthcare Scenario

The IEEE defines interoperability as "*the ability of two or more systems or components to exchange information and to use the information that has been exchanged*" [70]. The exchange of information is known as technical interoperability, whereas the ability to use the transferred information is known as semantic interoperability. Two independent systems are able to interoperate if they implement a specific interface I for that purpose. However, as shown in Equation (2.1) the number of interfaces to implement grows approximately half the square of the number of systems N able to interoperate.

$$I = \frac{N(N-1)}{2} \tag{2.1}$$

To perform an effective communication, both sender and receiver of information must share a common "framework" that allows their communication. This is the reason why the use of standards is so important to achieve interoperabilty, specially in the medical domain where there is a huge number of concepts, and a big number of systems which must cooperate. Nowadays there are different available standards in the healthcare domain which can be organized in different categories:

- **Messaging and data exchange standards:** these standards allow the exchange of information between systems and organizations in a consistent way, because they contain specifications for the format, the data elements and their structure. Common examples in this group are the Health Level 7 (HL7) for the exchange, integration, sharing, and retrieval of electronic health information [74], Digital Imaging and Communications in Medicine (DICOM) for handling, storing, printing, and transmitting information in medical imaging [104], ISO/IEEE 11073 which enables communication between medical devices and with external computer systems [76].

- **Terminology standards:** these standards are concerned with terms and they provide specific codes for clinical concepts. Examples in this group are Systemized NOmenclature of MEDicine Clinical Terms (SNOMED CT) which provides a collection of clinical terms covering diseases, findings, procedures, microorganisms, substances, etc. [124], International Classification of Diseases (ICD) provides a system of diagnostic codes for classifying diseases [72], Logical Observation Identifiers Names and Codes (LOINC) a standard for identifying medical laboratory observations [97].

- **Document standards:** these standards are intended to specify an architecture for the exchange of Electronic Health Records (EHRs). This group includes the Clinical Document Architecture (CDA) a standard based on the eXtensible Markup Language (XML) intended to specify the encoding, structure and semantics of clinical documents [75], and the CEN/ISO EN13606 a standard to define a rigorous information architecture for communicating part or all of the EHRs of a single patient [73].

Moreover, there are different approaches to support the use of existing healthcare standards. An example is the Integrating the Healthcare Enterprise (IHE) initiative [71], which provides a framework that defines a set of specific implementations of established standards, named profiles, to coordinate integration goals between different healthcare institutions. Other example is the Continua Health Alliance [44], an international nonprofit industry organization with more than 200 member companies, aiming to provide an ecosystem of interoperable health devices and services.

Health Level 7 International [67] is one of the most important organizations involved in the development of healthcare standards, with approximately 500 corporate members. The name "Level 7" is a reference to the seventh layer of the OSI communication model, which indicates that HL7 standards are focused on the application layer. A widely adopted standard of this organization is HL7 v2.x which defines electronic messages to support hospital workflows. The first version of this standard was released in 1989 and over the years it has been refined to cover the requirements in different institutions and in different countries. The HL7 v2's philosophy is that newer versions of HL7 v2 should be backwards compatible with older versions of the standard. This means that new data elements and messages added to newer versions are marked as optional elements, which in fact makes more difficult its implementation. The drawbacks found in HL7 v2 lead the development of the new HL7 v3 standard which was firstly released in 2005. HL7 v3 defines the Reference Information Model (RIM), an essential part of the HL7 v3 development methodology representing a large pictorial of the clinical data from which all messages are derived.

Another standard developed by the HL7 organization is the Clinical Document Architecture Release 2 (CDA R2) [49]. CDA is a document markup standard that specifies the structure and semantics of a clinical document for the purpose of exchange. A CDA document is a defined and complete information object that can include text, image, sounds, and other multimedia content. CDA documents are encoded in XML and they derive their machine processable information from the RIM. Every CDA document has a header and a body. The header identifies and classifies the document and provides information on authentication, the encounter, the patient, and the involved providers. The body contains the clinical report, organized into sections whose narrative content can be encoded using standard vocabularies. The CDA R2 model is richly expressive, enabling the formal representation of clinical statements (such as observations, medication administrations, reverse events, etc.) such that they can be interpreted by a computer. On the other hand, CDA R2 offers a low bar for adoption, providing a mechanism for wrapping a non-XML document.

HL7 standards have been adopted in some research projects on pervasive healthcare. Mainly there are projects that implement HL7 messages for the communication between the different parts of the system, and systems that create a CDA report of the state of the patient. In [91] the authors present a prototype of body sensor networks for home monitoring that utilizes HL7 messages and the IEEE 1073 standard, known as the Medical Information Bus (MIB). Their prototype has four major parts, i) base station, data logger, sensors and LabVIEW routines, ii) a local database, iii) an HL7 communication tool and iv) automated or doctor driven analysis and feedback. Data collected by sensors are uploaded wirelessly using MIB via Bluetooth to a data logger worn by the patient. The data logger uploads the measurements, again using MIB via Bluetooth, to a base station running a LabVIEW program which stores the values to a local database. The HL7 communication module is a client/server application implemented using the Interfacewares' Chameleon software, which can send the information from the local database

using HL7 messages to an external application situated in the hospital side. The novelty of this work is that physicians can remotely make changes to the monitoring system based on data, for example changing the sample rate of an electrocardiogram (ECG), closing the loop of the monitoring system. However, authors claim this as a potential add-on to the system.

A similar system is presented in [94] but using a mobile device and a telemedicine data center. The mobile device is in charge to gather physiological data information such as blood sugar, blood pressure, ECG, temperature and oxygen saturation in blood. While the telemedicine data center analyzes the data based on thresholds. The communication of the physiological data between both parts of the system is done using the encapsulated HL7 standard EHR. The originality of this system resides in the fact that the telemedicine data center can eventually trigger a conference with the mobile if an alert is detected.

Inside the hospitals there are also pervasive systems like MobileMed [41] using HL7 v2.x messages to send information. The goal of MobiMed is to allow to different healthcare institutions to access clinical results about patients. Each hospital has an HL7 Message Server, which is in charge to send the laboratory results to Central Clinical DataBase (CCDB) by generating each time the appropriate HL7 message. The information stored in the CCDB can be accessed by the physicians by means of a Personal Digital Assistance (PDA), this way they can share patient clinical information with small time delays. The authors assume that the CCDB, as a central shared repository, would be managed by a government organization. This system introduces an original approach to share data between hospitals, however the centralized CCDB represents a single point of failure of the system.

The CDA standard has been used as well in different research projects. In [60] the authors report a Home Telecare System consisting of a patient database and a report system. The database stores parameters extracted from raw signals of vital signs, whereas the report system takes the data from the database to perform analysis on it. The report system first converts the information to XML format which in turn is used to generate a CDA report, and then analyzes data with the trends path and the alerts path. Trends path performs statistical analysis of data like mean and standard deviation, while the alerts path is an expert system based on ripple-down rules which generates warnings and alerts. The combination of an analysis system with the generation of the CDA report is the novelty of this work.

A smart home healthcare system is presented in [85]. In this system the data about different activities is collected through motion sensors, preprocessed using different algorithms (sensory based, video based, location tracking), and stored in XML format. Each activity includes information about type of activity, sensor information, name of the person, activity name, identification of the sensor location, and occurrence time of the activity. The originality of this work resides on the HL7 compliancy module which generates a CDA document based on the activities. This CDA document can be then transmitted to all registered healthcare systems with the smart home.

Koutkias et al. [88] propose a novel framework focused on medication treatment manager to

provide safety with respect the medication by coping with Adverse Drug Events (ADE). Their architecture is composed of two subsystems the patient site and the medical site. Patient site has a body area network with sensors measuring the blood pressure and the heart rate, and a Mobile Base Unit (MBU) which coordinates the sensor network and notifies the monitoring to the medical site. The medical site is in charge to store the sensed parameters at the patient site and to send to the MBU information related with the prescription such as treatment goals in terms of monitored signs, important ADE that may occur, and ADE detection patterns. XML is used for the communications between both subsystems. From the medical site to the patient site the drug prescription information is encoded using an own schema, and in the reverse channel the reports of the monitoring are provided by using the CDA.

Research efforts had also been done on providing custom EHRs. In [128] the authors propose a standard data object structure based on XML to intermediate among hospitals. They define all the components of the document and provide the algorithms to generate and process them. However, this approach does not look suitable to provide global interoperability between institutions.

All the reviewed systems provide interoperability by sending HL7 messages or generating CDA documents. However, most of them are prototypes which have not been evaluated with potential users, and none of them provides information about which HL7 messages have been used nor the structure of the generated CDA documents. In this thesis we aim to provide a methodology to create such documents, which is based on the combination of templates that are filled each time with the proper values. Moreover, since our system is based on rules for monitoring the patients we use JavaScript Object Notation (JSON), a standard for sending structured data through the web, to format such rules.

2.2 Scalability of Healthcare Systems

Scalability is a concept which connotes the ability of a system to process growing volumes of work gracefully, and/or to be susceptible to enlargement, both due to accommodate an increasing number of elements or objects [24]. Therefore, scalabillity is a desirable attribute for any network, system or process. In the case of a server application there are few design principles that can be used to scale it up [117]:

- **Divide and conquer:** the idea is to split the system into smaller subsystems that must deal with focused tasks. This approach includes also the replication of the system to process specific load coming from the same physical place.
- **Asynchrony:** this means that the system can schedule the tasks to do according to the available resources, rather than process the tasks in the same order that they arrive.
- **Encapsulation:** by applying this principle the systems' components are loosely coupled,

so there is little or no dependence between them. Ideally the components should not wait for the work of the other components to perform their tasks.

- **Concurrency:** this principle helps to scalability by ensuring that the maximum possible work is active at all times, and activating new resources when they are needed.
- **Parsimony:** means that the designer must design the application carefully, taking into account that each piece of code has a cost. If the designer is not economical on what he or she does the costs for the tasks to do can increase exponentially.

The scalability is not taken as a concern in most of the pervasive healthcare systems. In [103] the authors review several systems and they conclude that the foremost issue of these systems is the scalability. However, in the literature we can find few pervasive healthcare systems which specifically address the scalability challenge.

The eCAALXY [113] is a system designed for elder people people suffering from co-mordibity, to improve their quality of life and reduce mordibity and mortality of elders. The system is composed by three subsystems. First, the mobile monitoring system which is a Smart Garment embedded with sensors and an Electronic Control Unit controlling the sensors. Second, the home monitoring system composed by a Set-Top-Box which allows to use the television as an interactive tool to receive health education via videos, show the vital parameters, check the health agenda, and do videoconference with the doctors; a router which acts as a hub for the sensors to send the sensed data to the caretaker site; and an Intelligent Sensor System composed by a set of sensors deployed in the home. Third, the caretaker site composed by a server responsible for patient management, data visualization, health agenda and observation pattern management. The authors report that due to the centralized communication architecture, the server on the caretaker side receives several dozen of messages per user each second. Thus, the system has low capability in scaling. To solve this issue the authors purpose, as future work, an improved version of the architecture including a middle-layer between the data-acquisition components and the caretaker server to achieve: i) decentralized data acquisition, ii) early pipeline processing, and iii) migration to well known technologies capable of handling high data rates. Therefore, the authors purpose to scale up their system by applying the next three techniques: i) assign the same task to multiple elements, ii) do an early processing of the information, and iii) increase the computational power of the elements of the system.

The Artemis monitoring system [22] is designed to perform real-time analysis of data coming from sensors. According to the authors, the high-frequency analysis of the physiological data streams could lead to early detection of life-threatening conditions, and they apply it into an Intensive Care Unit for neonates. One of the design goals of the system is to be able to scale with the number of data streams and patients connected to the system. In order to be able to interface with a great number of different medical devices, Artemis employs a set of hardware and software from Capsule Tech Inc. Data acquired from sensors are forwarded to a DataCaptor terminal unit which converts the data streams to IP data streams. Then these data are forward to

a Capsule DataCaptor Interface server which can support up to 500 simultaneously connected devices. The authors purpose the use of multiple servers to achieve scalability, although they claim that a single server is sufficient for the deployment of the system.

The authors of [98] propose an architecture in which a healthcare institution is able to manage data collected by wireless sensor networks. In order to be able to collect and access large amounts of data the architecture is based on cloud storage. The use of the cloud implies some security challenges. The first is to ensure the confidentiality of data stored on the cloud with a fine-grained access control. This challenge is addressed using Attribute Based Encryption. The second one is the management of data which is addressed by using an external institution called Healthcare Authority, in charge to enforce the security polices of the healthcare institution. The use of the cloud provides scalability due to its virtually infinite storage capacity, so in this work the idea to achieve scalability is to offer much more space than the one that will be needed.

Aingeru [130] is a tele assistance system for the elderly. Each monitored patient carries a PDA that is connected to sensors that sample physiological parameters. In the PDA there is an agent deployed with the Java Agent DEvelopment Framework (JADE) that analyzes the data forwarded by the sensors. Two types of alarms can be activated from the PDA, i) manual activation: when the patient feels bad can notify it, and ii) automatic activation: when the agent detects an anomalous situation an alarm is sent to the doctor. In the hospital side there is a server that stores in a database all data sent by patients. The relevant contribution of this paper is the local analysis of data done in the PDA. This measure reduces the amount of data sent through the network and therefore minimizes the amount of work that the server side must perform. This is demonstrated in the evaluation of the system where the communication costs are measured, i.e. the amount of data sent from the PDA to the server, comparing the approach of making local data analysis with other approaches where data analysis is done externally. The idea to provide scalability is to analyze data before using the network interfaces that in turn implies savings in battery consumption.

The pervasive healthcare system proposed in [51] is a solution for monitoring the activities of their users. This could be useful in measuring postoperative patient recovery. The system combines two types of sensors, wearable and ambient sensors. The wearable sensors are embedded in a device worn in the ear, whereas the ambient sensors are visual-based sensors. The system has a three-tier architecture. First, the sensing environment of the patient in which health data is captured and sent to a broker server through a gateway. Second, the data fusion and analysis layer composed by different distributed servers and brokers. The servers are in charge of performing different tasks such as acquisition, processing, storage and visualization of data, while the brokers provide communications between the servers. Third, the stakeholders interface, which allows to retrieve stored information or carry out specific data processing algorithms. The scalability of the system is based on the loose coupling of the different heterogeneous components of the system.

Other approach to achieve scalability is the one proposed in [86]. The infrastructure provided

in this work, although is not a pervasive healthcare system, allows to exchange EHRs between communities. Its purpose is to address the challenges not covered by the IHE Cross Community Access, a profile that defines a protocol to query and retrieve patient healthcare data across communities. One of these challenges is the scalability for which the system proposes the use of a peer-to-peer (P2P) network. However, using a fully decentralized P2P network the search of the desired information may require to flood the network. Instead of that, the architecture is based on the superpeer model on which there are special peers called superpeers. The superpeers provide directory services for their connected peers, and use its directory of indexes to route requests. In this work the techniques applied to achieve scalability are the delegation and/or division of tasks.

In the reviewed systems the challenge of achieving scalability was taken into account from the design stage. Only in the eCAALXY [113] the solutions are proposed after the evaluation of the system. All the above-mentioned systems offer different approaches to achieve scalability according to its specific application. However, most of the authors just claim their systems as being scalable systems without providing an idea on how scalable their system is. Only in the Aingeru [130] agent-based system the scalability is evaluated in terms of communication costs in a lab-defined scenario. In this thesis we follow a similar approach to achieve scalability, that is running the agents in charge of the monitoring task in the Tier 2 of PHSs. However, our work differs from Aingeru in different aspects. The MAGPIE agent platform is a reusable software solution that can be used to develop different PHSs, while Aingeru has been thought as a system that solves a specific problem. Our agent platform is designed to run on Android devices. Thus, adapted to the current mobile landscape.

2.2.1 Agent Platforms for Android

In the line of MAGPIE, different works already targeted Android to build an agent platform on top of it. These works are: the Agent Platform Independent Model (APIM) [4], JADE Android [135], JaCa-Android [120], Micro-agents On Android (MOA) [58], and Jadex [111].

The APIM is not an agent platform itself, but an agent model based on the commonalities of different agent models. The key components of this model are based directly on Android components, e.g. an agent is an extension of an Android service. An agent can have different *behaviors* that are used to distinguish between different environments. Each behavior groups a set of *capabilities*, which are tasks that an agent knows how to fulfill. As example, a chat behavior can have two capabilities: send messages and receive messages.

JADE Android, is the Android version of JADE-LEAP, the mobile version of the popular JADE agent platform [18]. This platform integrates Android with the JADE infrastructure, and it relies on running a distributed container between the Android smartphone and the JADE desktop version. This distributed container has features like the delegation of tasks to the back-end, so that the front-end becomes lightweight; and the handling of the exchange of information by the back-end, so that the front end can save energy. However, this split container only allows to run

one single JADE agent.

JaCa-Android is the porting to Android of JaCa, which integrates the two agent technologies Jason [26] and CArtAgO [116]. Jason is an agent programming language based on the Belief-Desire-Intention (BDI) model. The agents in JaCa have a reactive behavior and run continuously, reacting to the changes in the environment. CArtAgo is the framework to program and execute the environments. It is based on the Agents and Artifacts model, where an artifact is an exploitable infrastructure that the agents can instantiate, share and use. The porting of JaCa to Android relies on a set of provided artifacts designed for exploiting specific Android functionalities like SMS or the GPS sensor.

MOA relies on an organizational architecture for agents, where agents can play different roles and each role can subscribe to a particular set of events (like the initialization of a new agent). The communications between agents are handled by means of μ-intents, that is an abstraction to model the request of execution of a particular task. The framework is divided in three layers: i) the agent logic layer that models the artifacts used in the application; ii) the message routing layer where the μ-intents, and the roles subscribed to each event are registered; iii) meesage transport layer, which acts as an agent management system where all agents are registered.

The Jadex agent platform has also been ported to Android. Jadex was initially developed as a BDI layer on top of JADE, but currently supports also the development of micro agents. These agents are simple and reactive agents designed to be fast and lightweight. Micro agents are programmed as Java objects by extending a framework class, which has a set of predefined methods for their initialization, logic, and message handling.

Though the existing agent platforms for Android offer quite complete functionalities, they lack from features that we need in the context of PHSs. In this scenario where the goal is to monitor continuously a person, we need that agents react to the physiological measurements carried on by sensors. Thus, an abstraction layer modeling sensors is needed, so that agents can be decoupled from the implementation details of the particular sensors used by the patient. Another desired feature in this scenario is to be able to modify the agent's medical knowledge, to do it at run time, and by non-technical users like medical doctors. As explained in chapters 4 and 5, in MAGPIE this goal is achieved through the combination of Prolog agents, with a graphical rule editor that allows modeling medical knowledge in a high-level way.

2.3 Formalization Models in the Medical Domain

To formalize the medical knowledge with methods or algorithms in order to improve practitioner performance and patient outcomes is a challenging task due to the complex nature of biological systems. Besides that, the process of incorporating the studies produced in clinical research to the healthcare workflow is not an easy task. This is specially remarkable when different clinical studies are conflicting.

The integration of decision support models into a PHS is not mandatory requirement. We can find examples like the PHS presented in [59], where the main goal is to continuously monitor vital signs and send them to the hospital server-side by applying interoperability standards. Another similar example is the system reported in [134], which is focused on heart failure patients. This system provides to the patient educational material on heart failure in addition to the monitoring task. Though these systems consider aspects that give an added value, clinical practice can also be improved when considering decision support models [83]. Most of these models come from the fields of Artificial Intelligence (AI) and machine learning, and are not specific to the medical domain but also applied into other fields of engineering. A systematic review on decision support systems in medicine can be found in [119]. In here we focus on briefly describe the different available methods, and to see different examples of their use in the healthcare domain. We classify these methods into six different categories: supervised learning, unsupervised learning, probabilistic logic, fuzzy logic, ontological reasoning and temporal reasoning.

- **Supervised Learning:** the techniques lying in this category consist on inferring a function from labeled training examples, that we can later use with different data. These techniques have the advantage of having a mathematical foundation. On the other hand, they can require a significant amount of data to train the algorithm. *Decision tree* is a technique where a tree is build from the dataset and used to classify data. This technique has been used for activity recognition [62], and to suggest diet and physical activity [105]. *Artificial neural networks* is a model that tries to mimic a biological brain that consists on a large number of small interconnected neurons. This technique has been used for the prediction of heart attacks [114] and the recognition of motions in rehabilitation exercises [93]. *Support vector machines* are used to recognize patterns in data that can be used for classification and regression. Examples of their use are the recognition of a patient state like pain, tensed, etc. from the speech and face images [69], and the diagnose of obstructive sleep apnea [38].

- **Unsupevised Learning:** this family of techniques attempt to find hidden structures in unlabeled data. In contrast with the supervised techniques, in unsupervised learning there is no error or reward signal to evaluate a potential solution as there is no used training data. This fact makes these techniques difficult to validate. There are techniques like *clustering* that consist on different methods to group unlabelled data in subsets named clusters, so that the data within a cluster have very high similarity in comparison with the data from the other clusters. *K-means* is a clustering algorithm that has been used to categorize walking-ages for a group of people [77]. Unsupervised techniques based on neural networks such as *self-organizing maps* are used to recognize activities from video sequences [56]. Blind source separation techniques like *principal component analysis* have been used to extract ECG features for studying cardiac effects of the diabetes [80].

- **Probabilistic Logic:** are a set of techniques that combine deductive logic with probabilities that are attached to facts related to the problem. Although these methods can handle uncertainty, they can only be applied when the probabilities are known. They are typically used to help in the diagnose of diseases and activity recognition. Methods like *bayesian networks* are combined with graphical models to represent events and the relationship between them. Bayesian networks have been used to diagnose diseases like depression [37] and cardiovascular disease [121], and are used in [138] as the intelligent part of a PHS to diagnose preeclampsia. *Hidden Markov Models* are used for the early detection of bradycardia episodes in infants [100]. *Markov Logic Networks* have been used for activity recognition of dementia patients [61]. *Dempster-Shafer theory* is used for sensor data fusion in a fall detection system [35].

- **Fuzzy Logic:** allows reasoning that is not crisp, so that partial truth values that range from 0 to 1 are acceptable. This contrasts with traditional logic where the values can be either true or false. Fuzzy logic is similar to probabilistic logic, it also handles uncertainty but confidence values represent degrees of membership instead of probabilities. This allows to represent real world scenarios in a more natural way. Fuzzy rules have been used to detect anomalies in a smart home environment [145], to advice recommendations to patients and practitioners in a PHS [19], and to diagnose hypertension risk in young adults [2], among others.

- **Ontological Reasoning:** is based on the use of ontologies as a formalism for representing the medical domain, in combination with semantic web query languages to retrieve and derive implicit ontological knowledge. A common semantic web language used for ontology representation in medicine is Web Ontology Language (OWL) or variations of it. In [79] a monitoring system for diabetes is presented, where OWL is used to model a diabetic patient profile and SWRL rules to infer new properties about the patients. A similar system is presented in [101], where mobile devices are used for cardiac monitoring. In this system Jena rules are used for asserting new facts and OWL to model the patient summary. In the context of rehabilitation, a multiagent system is presented in [126] where rehabilitation concepts are modeled in OWL DL and SPARQL is used as a query language. Modeling knowledge through ontologies can also be combined with other types of reasoning like bayesian networks [8].

- **Temporal Reasoning:** time-related issues for processing information in medicine combines knowledge from different fields like philosophy, logic, AI, computational linguistics, and biomedical informatics. In general terms, the temporal notion involves the definition of object's properties that define a certain state, which changes by the occurrence of events. Temporal reasoning in medicine can be subclassified in different ways. In [43] temporal-based systems are classified from the application point of view (e.g. diagnosis) and clinical areas (e.g. cardiology), and from the methodologies and theories used. In [10]

the issues about representing and retrieving temporal data are discussed, and different systems are surveyed according to different stages of patient care: diagnosis, prognosis and treatment. Another survey with special emphasis on natural language processing classifies the literature in three different categories: category 1 applies theories and models from the temporal reasoning in AI, category 2 are frameworks the meet the needs from clinical applications, and category 3 deals with issues such as temporal granularity and temporal uncertainty [149].

In this thesis we model the medical knowledge following a temporal reasoning approach that is based on the Event Calculus (EC) [89]. Previous works already applied EC to different healthcare-related scenarios. One of the early attempts proposed a prototype for the management of mechanical ventilation, which uses an efficient variation of the EC [40]. More recent applications include conflict detection in telecare services [23], a PHS for the management of the Diabetes Mellitus (DM) [78], event recognition in Ambient Assisted Living (AAL) [82], and operating robots in a rescue scenario [63]. The focus of this thesis have common points with respect the formalization of the medical knowledge in the PHS presented in [78]. In this system, like in ours, the knowledge is modeled through EC clauses that form a set of monitoring rules, which are used to provide feedback on the status of the patient. Our work focuses on extending this approach by providing an application where medical doctors can define such rules in a graphical way, since they are not supposed to be familiar with knowledge on logic programming to define the rules. Moreover, the monitoring rules are integrated with the MAGPIE agent platform, which is our approach through scalable PHS.

We apply the methods of this thesis on the use case of DM management. A different approach to manage this disease is referred as the artificial pancreas [42]. This approach combines three different elements to emulate the functionality of a real pancreas: a Continuous Glucose Monitoring (CGM) device, an insulin pump and an intelligent algorithm. Within this context several algorithms have been proposed to predict blood glucose levels in a given time horizon, in order to calculate the right dose of insulin to be injected by the insulin pump. These prediction algorithms are based on different of the above-mentioned techniques like rules [33], artificial neural networks [109], proportional integral derivative controllers [107], and neuro-fuzzy techniques [146]. There are also algorithms focusing on denoising CGM readings [55] that can help on improving the prediction. The artificial pancreas as a method to manage DM has the drawback that insulin pumps are invasive devices which can cause skin infection and dermatological changes at the site of infusion [131], and problems can occur with blocked, kinked or leaking cannulas [68]. Moreover, pumps do not send electronically to the doctor injected insulin doses, so that the possibility to adapt the treatment is limited by the times that the doctor can visit the patient. Another issue with respect CGM devices is that they provide glucose values measured in the interstitial fluid so there is a delay of about ten minutes between the measurement and the plasma glucose [84].

Chapter 3

Interoperability: Electronic Health Records for Personal Health Systems

Contents

This chapter describes the strategy to provide interoperability to our Personal Health System (PHS), which consists on the use of Clinical Document Architecture (CDA) documents flowing from the Tier 2 to the Tier 3 of the PHS. These CDA documents are generated in order to report the physiological values and alerts that capture the status of the patient at a particular point in time. Section 3.1 gives an overview on the composition of these CDA documents, and how they are generated. In Section 3.2 and Section 3.3 the different elements conforming the header and the body of the generated documents are specified. In Section 3.4 the generation of the documents is evaluated in terms of two different generation strategies: a strategy based on generating a CDA document every time that an alert is triggered, and a strategy based on generating a CDA document at the end of the day. The results highlights the advantages on generating the documents according to the alerts.

3.1 CDA Documents for GDM

The aim of our PHS is to help on the management of Diabetes Mellitus (DM). In this chapter we focus on Gestational Diabetes Mellitus (GDM), a type of diabetes affecting around 5% of pregnant women due to an increased resistance to insulin [81]. GDM can increase the risk of health problems developing in an unborn baby, so it is important that the glycemia levels of the pregnant woman are under control.

The Tier 2 of our PHS consists on an Android application in which the patient can enter a series of health data related with GDM. These data are divided in three main categories that are physiological parameters, symptoms and medications. Table 3.1 shows the elements that can be reported in each category. The application has other functionalities like analyzing the data and reporting alerts according to a set of GDM-related monitoring rules, the GDM patient can also check and eventually correct the data that she has entered. All data are stored encrypted in the phone and sent to the Tier 3 formatted according to the CDA standard, when the Android phone has network connectivity.

Table 3.1: Health data related with GDM encoded in the body of the CDA document

Physiological Parameters	**Symptoms**	**Medications**
Blood pressure	Chest pain	Insulatard
Heart rate	Edema	Huminsulin basale
Blood sugar	Dyspnea	Levemir
Weight	Blurred vision	Novorapid
	Headache	Humalog
	Epigastric pain	Metformin

In addition to the health information reported in Table 3.1, we added another category to the CDA documents related with the alerts produced by the application. The CDA documents are generated and sent when a new alert is triggered. In order to accomplish this task, the application's source code contains a package with all the necessary Java classes and resources to create these CDA documents. Figure 3.1 shows a UML diagram of the classes involved in this process. The `CDADocument` class corresponds to the clinical document intended to send, and it contains all the necessary methods to build it. In addition, each one of the rest of the classes is linked to an eXtensible Markup Language (XML) template file already formatted according to its representation in the CDA standard. When building a new clinical document, all the necessary XML templates are collected, their variable values and attributes are selected using XPath expressions, and the missing values are filled with its corresponding string representation. The complete set of XML templates used as well as their associated XPath expressions are specified in Appendix A.

One of the properties of CDA documents is that they are persistent in nature and maintained

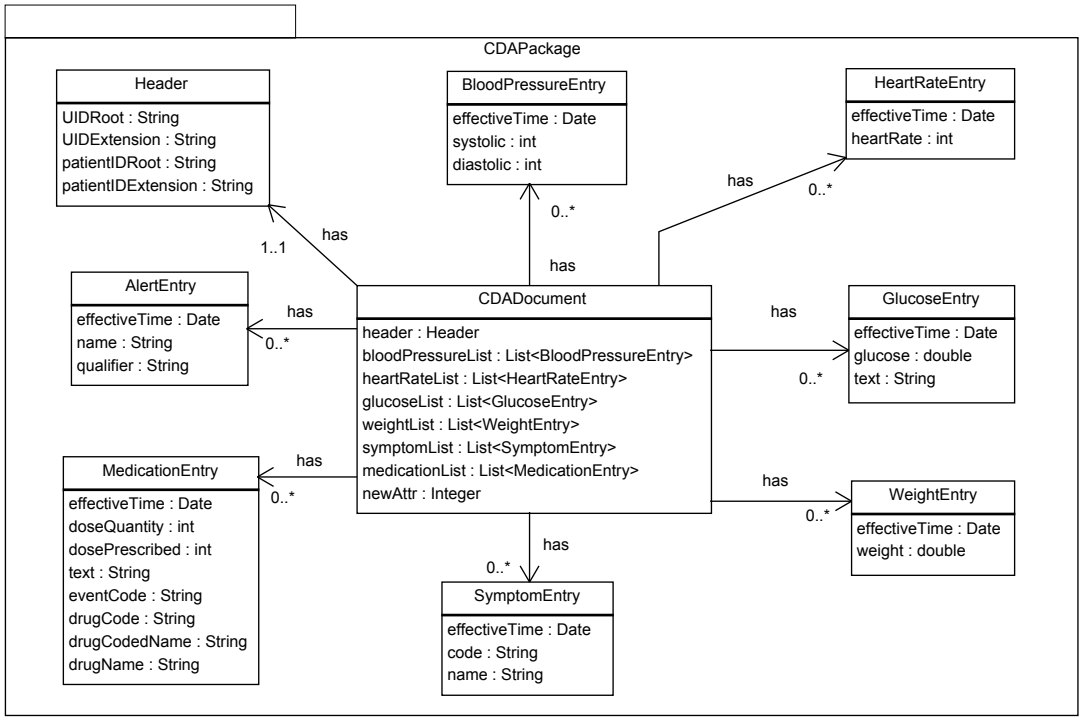


Figure 3.1: UML diagram of the classes involved in the generation of the CDA documents

by an organization entrusted with its care [25]. To accomplish this requirement we use BaseX [16] in the Tier 3 of the system, a XML database that allows to store the clinical documents in its original format. This choice also implies the use of XQuery to query information on the CDA documents. In the next sections we explain the structure of each part of the CDA documents specifically generated for GDM.

3.2 CDA Header

The header part of the generated CDA documents contains only the mandatory elements required by the standard. The optional elements are not used in order to minimize the amount of data sent by the application through the network interface of the mobile phone.

Every header part in a CDA establishes the default context for the contents of the clinical document. The set of minimum mandatory elements to set this context includes the following:

- The identification of the document, which is defined with the `<id>` element. This XML element has two attributes: root and extension. The root attribute identifies the universe of the clinical document, and the extension provides the uniquely identification for the clinical document.

- The type of document, specified with the <code> element. We use the Logical Observation Identifiers Names and Codes (LOINC) code "51855-5" with name "Patient Note", as it describes well the type of clinical document we want to generate. The term definition of this code states "*A patient authored note is generated by a patient, or a patient agent, acting in a non-clinical role to provide clinically relevant information*" [97].
- The creation time of the document, defined with the <effectiveTime> element. This element is specified in the following format: yyyyMMddhhmm.
- The confidentiality of the document, defined with the <confidentialityCode> element.
- The patient (or patients) whose document belongs to. All the patients' information is inside the <recordTarget> element. It can include the name, gender, address and other information, but the only mandatory field is its identification specified with an <id> element.
- The author of the document, can be someone or some device with the role of author. The author is defined with the <author> element, and its child <assignedAuthor>. Again, the author can be specified by providing its name, address, phone, email, but the only mandatory field is its identification specified with the <id> element. In our case the patient and the author of the document are the same person.
- The organization that is in charge of maintaining the document. This element is defined by the <custodian> XML tag. Although the name, address, telephone and other information about the organization can be specified, the only mandatory element is the <id> element.

3.3 CDA Body

The body part of the CDA documents for GDM is an XML structured body divided in four different <component> elements, each one with one <section> element. Each section encodes one of the following groups: physiological parameters, symptoms, medications and alerts. In addition, each section has <entry> elements encoding the medical information reported on Table 3.1. Furthermore, the sections are identified by a LOINC code through the code and display name attributes of the <code> element (Table 3.2). Other standard vocabularies used in the body are Systemized NOmenclature of MEDicine Clinical Terms (SNOMED CT) to encode the physiological parameters and the alerts, International Classification of Diseases (ICD)-10 for symptoms, and Anatomical Therapeutic Chemical classification system (ATC) for medication.

3.3.1 Physiological Parameters

The section corresponding to the physiological parameters can have four different kind of entries wrapped by <entry> elements, each one coding a different physiological parameter. These

Table 3.2: LOINC codes used to identify each section of the body

Section	Code	Display name
Physiological Parameters	8716-3	Vital signs
Symptoms	10164-2	History of present illness
Medications	10160-0	History of medication
Alerts	74018-3	Alert

parameters can be the blood pressure, the heart rate, the blood sugar, or the weight. All of these physiological parameters are encoded as observations using the `<observation>` element. An observation is an act which can be though as a "non-altering" procedure that results in a value [25]. In the case of this section a value is a physical quantity of a physiological parameter, although it can be virtually anything.

Each physiological parameter is identified by its corresponding SNOMED CT code using the `<code>` element (Table 3.3), specifies its measure units in the unit attribute of the `<value>` element, and has associated metadata such as the time of the measurement specified in the value attribute of the `<effectiveTime>` element. Figure 3.2 shows the encoding of the blood pressure. The class code attribute of the `<observation>` element defines the kind of the act that is, while the mood code attribute describes its placement in time. In this case the value of the mood code of all physiological parameters is "EVN" as it defines an act that has already occurred.

Table 3.3: SNOMED CT codes used to identify the physiological parameters

Physiological Parameter	Code	Display name
Blood pressure	251076008	Cuff blood pressure
• Systolic	271649006	Systolic BP
• Diastolic	271650006	Diastolic BP
Heart rate	364075005	Heart rate
Blood sugar	302789003	Capillary blood glucose measurement (procedure)
Weight	363808001	Body weight measure

The encoding of the blood pressure differs from the other physiological parameters as it consists of two different parameters, the systolic and the diastolic blood pressure. This relationship is encoded with two different `<entryRelationship>` elements inside the `<observation>` element. The encoding of the heart rate also differs from the other physiological parameters as it is measured in beats per minute. This fact is expressed with a `<denominator>` element which is a child of the `<value>` element.

```
<observation classCode="OBS" moodCode="EVN">
    <code code="251076008" codeSystem="2.16.840.1.113883.6.96"
    codeSystemName="SNOMED CT" displayName="Cuff blood pressure"/>
    <effectiveTime value="201301221746"/>
    <entryRelationship typeCode="COMP">
        <observation classCode="OBS" moodCode="EVN">
            <code code="271649006" codeSystem="2.16.840.1.113883.6.96"
            codeSystemName="SNOMED CT" displayName="Systolic BP"/>
            <value unit="mm[Hg]" value="120" xsi:type="PQ"/>
        </observation>
    </entryRelationship>
    <entryRelationship typeCode="COMP">
        <observation classCode="OBS" moodCode="EVN">
            <code code="271650006" codeSystem="2.16.840.1.113883.6.96"
            codeSystemName="SNOMED CT" displayName="Diastolic BP"/>
            <value unit="mm[Hg]" value="80" xsi:type="PQ"/>
        </observation>
    </entryRelationship>
</observation>
```

Figure 3.2: Blood pressure measurement encoded in the CDA

3.3.2 Symptoms

The section corresponding to the symptoms can have six different kind of entries, each one coding a different symptom. The symptoms, as the physiological parameters, are encoded as observations with the `<observation>` element. The identification of the symptom is done using the ICD-10 vocabulary (Table 3.4 specifies the codes used). In each symptom entry the child `<code>` element of the `<observation>` provides the identification of the symptom. Every symptom has associated metadata corresponding to the time in which the symptom occurred. Figure 3.3 shows an example encoding the headache symptom.

Table 3.4: ICD-10 codes used to identify the symptoms

Symptom	Code	Display name
Chest pain	R07.4	Chest pain, unspecified
Edema	O12.0	Gestational oedema
Dyspnea	R06.0	Dyspnoea
Blurred vision	H53.8	Other visual disturbances
Headache	R51	Headache
Epigastric pain	R10.1	Pain localized to upper abdomen

```
<observation classCode="OBS" moodCode="EVN">
    <code code="R51" codeSystem="2.16.840.1.113883.6.3"
    codeSystemName="ICD10" displayName="Headache"/>
    <effectiveTime value="201305212115"/>
</observation>
```

Figure 3.3: Headache symptom encoded in the CDA

3.3.3 Medications

The section corresponding to the medications can have six different kind of entries, each one coding a different medication. All the medications of this section are types of insulin because of the target disease of the PHS. In this section the medications are encoded using the `<substanceAdministration>` element. This element is intended to represent the administration of a particular substance, e.g. a medication, immunization or other substance to a patient [25].

Each medication is identified using its ATC code (Table 3.5). In addition the `<name>` element provides the name of the medication as it appears in the mobile application. Figure 3.4 shows an example of the encoding of one drug. The metadata associated with the medication entry are: i) an optional comment related with the entry that the patient can write into the mobile application wrapped with the `<text>` element, ii) the time when the medication was taken encoded with two `<effectiveTime>` elements, and iii) the dose amount taken by the patient and the dose amount prescribed by the doctor, both expressed as insulin units.

Table 3.5: ATC codes used to identify the medications

Medication	Code	Display name
Insulatard	A10AC01	insulin (human)
Huminsulin basale	A10AD01	insulin (human)
Levemir	A10AE05	insulin detemir
Novorapid	A10AB05	insulin aspart
Humalog	A10AB04	insulin lispro
Metformin	A10BA02	metformin

In the mobile application the time at which a medication was taken is specified with two elements: a time stamp, and a period specifying when was the dose taken with respect the meals e.g. before breakfast, after breakfast, etc. Besides, the `<substanceAdministration>` element of the CDA specifies the dose frequency with `<effectiveTime>` elements using the General Timing Specification (GTS) data type. The GTS data type allows to express complex timings as a set of time intervals, using different kind of operations such as intersections, unions and differences. Thus, we specify the time of administration of a medication as the intersection of

```
<substanceAdministration classCode="SBADM" moodCode="EVN">
    <text>optional comment related with the entry</text>
    <effectiveTime xsi:type="TS" value="201305211250"/>
    <effectiveTime xsi:type="EIVL" operator="A">
        <event code="ACD"/>
    </effectiveTime>
    <doseQuantity value="2.5" unit="IU"/>
    <dosePrescribed value="2" unit="IU"/>
    <consumable>
        <manufacturedProduct>
            <manufacturedLabeledDrug>
                <code code="A10AE05" codeSystem="2.16.840.1.113883.6.73"
                codeSystemName="ATC" displayName="insulin detemir"/>
                <name>Levemir</name>
            </manufacturedLabeledDrug>
       </manufacturedProduct>
    </consumable>
</substanceAdministration>
```

Figure 3.4: Levemir medication taken before lunch encoded in the CDA

a Time Stamp data type and a Event-related periodic InterVaL of time (EIVL) data type. The EIVL data type is used to represent events that are tied to meals and sleeping. The `<event>` element is used to specify the specific event with its code attribute. The codes that can be used are fixed by the Health Level 7 (HL7) standard. Table 3.6 shows the ones we have used in our application.

Table 3.6: Event related timing codes used to identify times of the day

Time of the day	Code	Meaning (from Latin)
Before breakfast	ACM	ante cibus matutinus
After breakfast	PCM	post cibus matutinus
Before lunch	ACD	ante cibus diurnus
After lunch	PCD	post cibus diurnus
Before dinner	ACV	ante cibus vespertinus
After dinner	PCV	post cibus vespertinus
Later	ICV	inter cibus vespertinus

The `<substanceAdministration>` element only allows the codification of one dose. However, in the mobile application the patient can type both, the insulin dose prescribed by the doctor and the insulin dose really injected. This is because diabetes is a self-managed disease, so the patient has certain degree of autonomy in deciding which is the right dose of insulin she needs, as the blood sugar levels depends on the type and amount of meals taken. To encode the insulin

dose taken by the patient we use the `<doseQuantity>` element defined by the standard, and we added the `<dosePrescribed>` element to encode the dose prescribed by the doctor. This is the only XML element we have defined in the whole CDA in order to be able to encode all the health-related data specified into the mobile application. The addition of locally defined XML elements is something allowed by the standard as in the Section 1.4 it states "*Locally-defined markup may be used when local semantics have no corresponding representation in the CDA specification.*"

The addition of the `<dosePrescribed>` element has some implications with respect the interoporability with other systems. In particular, the validation of a CDA containing extensions must be done in stages [25]. The first stage should validate the extension content, by using W3C Schema or ISO Schematron that must be provided to the rest of applications. Once the extensions are validated, these must be removed before other validations occur. This procedure can be done using a XSLT stylesheet.

3.3.4 Alerts

The section for the alerts can have four different kind of entries, which are mapped to six different monitoring rules. These monitoring rules are based on glucose readings during the pregnancy and were defined by the medical doctors involved in the study. The alerts and their associated monitoring rules are the following,

- **Alert 1:** Hypoglycemia.
  - **Rule 1:** There are two consecutive days with glucose values less than 4 mmol/L in the same period of the day.
- **Alert 2:** Severe hypoglycemia.
  - **Rule 2:** Two consecutive glucose values are less than 4 mmol/L within one hour.
- **Alert 3:** Postprandial hyperglycemia.
  - **Rule 3a:** Two times during the last four preceding days the glucose value is bigger or equal to 8 mmol/L in the periods after the meals.
  - **Rule 3b:** Three times during the last week the glucose value is bigger or equal to 7 mmol/L in the periods after the meals.
- **Alert 4:** Fasting hyperglycemia.
  - **Rule 4a:** Two times during the last four preceding days the glucose value is bigger or equal to 5.8 mmol/L in the same periods before the meals.
  - **Rule 4b:** Three times during the last week the glucose value is bigger or equal to 5.3 mmol/L in the same periods before the meals.

```
<observation classCode="OBS" moodCode="EVN">
    <code code="302866003" codeSystem="2.16.840.1.113883.6.96"
    codeSystemName="SNOMED CT" displayName="Hypoglycemia (disorder)"/>
    <effectiveTime value="201305210745"/>
    <value xsi:type="CD" code="24863003"
    codeSystem="2.16.840.1.113883.6.96" codeSystemName="SNOMED CT"
    displayName="Postprandial (qualifier value)"/>
</observation>
```

Figure 3.5: Postprandial hypoglycemia alert encoded in the CDA

The four above mentioned alerts are encoded in the CDA documents as observations. Three different child elements encode the semantics of an alert (Figure 3.5). The `<effectiveTime>` element specifies the point in time when the alert was fired. The `<code>` element encodes the type of alert, i.e. hypoglycemia or hyperglycemia, and the `<value>` element encodes a qualifier that accompanies the alert, i.e. severe, postprandial or fasting. Both, alerts and qualifiers, are specified through the SNOMED CT codes shown in Table 3.7.

Table 3.7: SNOMED CT codes used to identify the alerts and their qualifiers

		Code	Display name
Alert	Hypoglycemia	302866003	Hypoglycemia (disorder)
	Hyperglycemia	80394007	Hyperglycemia (disorder)
Qualifier	Severe	24484000	Severe (severity modifier) (qualifier value)
	Postprandial	24863003	Postprandial (qualifier value)
	Fasting	16985007	Fasting (finding)

3.4 Evaluation

The evaluation of the impact of providing interoperability to the PHS has been performed in terms of data sent through the network from the Tier 2 to the Tier 3 of the system. Two different strategies for sending the CDA documents have been considered, which are the following:

- **Alert strategy:** a CDA document is generated and sent when an alert is triggered, or alternatively if there is a period of one week without having triggered any alert. This last condition avoids to not notifying the doctor during long periods of time.

- **Daily startegy:** a CDA document is generated and sent at the end of every day. These documents do not contain alerts, as in this strategy alerts are computed in Tier 3.

The evaluation consists on a retrospective analysis in the MONDAINE dataset. This dataset includes 12 patients diagnosed with GDM between the 24th and the 32nd gestational week and treated at the Lausanne University Hospital. The patients were managed using a PHS to report glucose, insulin and symptoms, and the medical stuff in charge of the patients were notified with the previously reported alerts. The system was combined with regularly scheduled clinic visits. The clinical trial consisted on reporting the capillary blood glucose values seven times per day, before and after each meal, and one measurement two hours before going to sleep. The patients had also to report any symptoms, and the patients under insulin treatment the insulin doses as well.

Table 3.8 summarizes the statistics of the dataset. Each patient used the PHS during a period between five and seven weeks. The column regarding the number of glucose samples per day (3rd column) gives an idea on how well each patient followed the trial, whose value should be 7±0. The columns regarding the maximum and minimum glucose measurements (5th and 6th columns) gives an idea on which alerts can be triggered by each patient. As example, patients with a minimum glucose value bigger than 4 mmol/L should not trigger the two types of hypoglycemia alerts (alerts 1 and 2). Four patients received insulin treatment. In particular, patients 3 and 6 received basal insulin and patients 5 and 9 received insulin bolus. The column corresponding to the total number of symptoms shows that most of the patients were not reporting their symptoms.

Table 3.8: Statistics describing the MONDAINE dataset, which consists on 12 GDM patients (glucose measurements in mmol/L, insulin doses in IU)

Pat.	Days	Glucose				Insulin		Symptoms
		# per Day ($\bar{x}$±SD)	Values			# per Day ($\bar{x}$±SD)	Values ($\bar{x}$±SD)	# Samples
			$\bar{x}$±SD	Max.	Min.			
1	42	3.1±1.10	5.35±0.81	7.7	4.2	-	-	-
2	64	3.8±0.42	5.68±1.20	9.9	3.8	-	-	-
3	77	4.8±0.42	5.23±1.04	7.2	3.7	2.1±0.99	7.79±3.07	5
4	67	3.8±0.42	5.50±1.17	9.1	3.7	-	-	-
5	58	2.6±0.52	5.83±1.32	10.4	4.0	1.0±0.00	26.29±6.88	-
6	42	3.5±1.08	5.78±1.66	16.4	3.4	2.7±0.95	9.00±0.00	-
7	71	4.1±0.32	5.13±0.98	7.5	3.7	-	-	-
8	36	3.7±0.95	6.01±0.81	7.5	4.4	-	-	1
9	60	2.2±0.79	5.84±0.95	8.7	4.3	1.1±0.32	21.18±4.12	-
10	73	3.6±0.70	5.64±1.20	9.2	3.7	-	-	-
11	64	4.1±0.57	5.52±0.90	8.4	4.1	-	-	-
12	56	3.9±0.57	5.59±1.07	8.8	3.9	-	-	-

The metrics considered in this evaluation are the total number of documents generated, and

the total bytes sent through the network. To compute the size that would have each document, the size of the different document parts has been aggregated (Table 3.9).

Table 3.9: Size in bytes of the different document parts

	Part	Size (bytes)
	Header	1739
Sections	Vital signs	165
	Symptoms	181
	Medications	170
	Alerts	160
Entries	Glucose	404
	Symptom	237
	Medication	678
	Alert	420

Table 3.10 shows the different alerts that were reported by each patient, and the results comparing the both strategies defined previously. Notice that patient 1 do not trigger any of the defined alerts, however it generates six documents with the alert strategy. This is due to the fact that with this strategy documents are also generated at the end of a period of seven days without having reported any alert. In terms of CDA documents generated, the alert-based strategy generates 200 documents, while in the daily-based strategy this value is 711. This difference represents an increment of approximately 3.5 times. Comparing the total bytes sent through the network, the increment is about 87%. This difference in the total bytes sent is due to the fact that a CDA document always has a header that in our case has a considerable wight in comparison with the rest of the elements of the document. These results also suggest that with the alert strategy there are savings in battery consumption due to less data sent, something that is relevant when working with mobile devices.

Table 3.10: Alerts reported per patient and results for both strategies

Patient	# Alerts						Alert Strategy		Daily Strategy	
	R1	R2	R3a	R3b	R4a	R4b	# CDA	Total bytes	# CDA	Total bytes
1	-	-	-	-	-	-	6	55864	42	148768
2	-	-	-	-	2	17	21	143924	64	259168
3	-	-	1	2	2	21	26	218139	77	398259
4	2	-	-	-	-	15	20	142880	67	269044
5	-	-	7	14	4	9	35	199982	58	264408
6	-	-	-	-	1	5	9	141604	43	237073
7	1	-	-	-	-	3	12	128592	71	289484
8	-	-	-	-	-	4	8	76954	36	148826
9	-	-	3	9	1	3	21	139018	60	244872
10	3	-	-	-	3	9	21	165844	73	302963
11	-	-	2	-	-	-	10	107868	64	250280
12	-	-	-	-	-	5	11	101008	56	224348
						TOTAL	200	1621677	711	3037466

Chapter 4

Scalability: The MAGPIE Agent Platform

Contents

The previous chapter focused on how interoperability is provided with the use of the Clinical Document Architecture (CDA) standard into an intelligent mobile Android application used as Tier 2 of our Personal Health System (PHS). This chapter introduces the MAGPIE agent platform, which is the framework providing intelligence to that application, and our strategy to achieve scalable PHSs. Section 4.1 describes the context and the goals of this agent platform. The different layers conforming the architecture of the platform, and the different agents that can run on it are detailed in Section 4.2 and Section 4.3 respectively. Section 4.4 shows the interactions taken place between the MAGPIE components and the rest of the elements of the PHS. Section 4.5 evaluates the scalability of the proposed agent platform, by comparing the approach of running agents in Tier 2 against the approach of running the agents in Tier 3. The results highlights the advantages of distributing the computations into the mobile part of PHSs.

4.1 Landscape of an Android Based Agent Platform

In the context of PHSs the use of mobile devices in conjunction with sensors deployed on the body gives the vision of "*healthcare to anyone, anytime and anywhere*" [137]. In the recent years the market of smartphones and tablets has been well established. Nowadays the smartphone's hardware components offer powerful computation capabilities that allow to perform the same tasks we do with a desktop computer. Another factor that contributed to the establishment of this new scenario for mobile computing is the apparition of operating systems like Android [32] specially designed for handheld devices. Android as it is offered as an open source solution, can be used by different vendors in their products without adding additional costs. Moreover, application developers can create and publish applications for this operating system and target a wide range of devices. In the particular case of PHSs, there is a key fact in this new generation of mobile devices, which are the integrated sensors like accelerometers, GPS receiver, ambient light, etc. that can provide contextual information that complements the one provided by the sensors of the Body Area Network (BAN).

In this context for mobile computing, we developed the MAGPIE agent platform. This agent platform aims to help on the development of Android mobile applications with the goal of monitoring chronic diseases in PHSs. The use of agents in PHSs can simplify the modeling of medical knowledge as they are autonomous software entities that purse a set of goals [142] in an intelligent way by applying Artificial Intelligent reasoning techniques such as deduction, and act proactively, without necessarily receiving a stimulus from the user. This set of properties can benefit the current definition of PHSs, by having monitoring tools that are capable of reasoning in a complex and proactive way on the current patients' physiological parameters. Moreover, the deployment of the agents in the Tier 2, that is in the mobile device from each patient, aims to improve the scalability of these systems in comparison with the current state-of-the-art approach where the computations for monitoring the patient are all done in Tier 3. This last tier is shared by all the patients using a particular PHS and therefore represents an inherent bottleneck of the system.

The MAGPIE agent platform is based on the concept of agent environment as a first class abstraction [141]. The agent environment concept, is becoming increasingly more important to simplify the definition and deployment of multiagent applications, by mediating the interaction between the agents and resources deployed in the system, by hiding to the agents the complexity of dealing with the state of resources external to the agent, and by providing standard interfaces and standard descriptions to resources so that the agents can utilize them to achieve their own goals.

In summary, MAGPIE has been designed for the current mobile landscape, with the aim of tackle technological challenges arising from the development of PHSs like their scalability. But also how the medical domain knowledge is modeled through the use of agents to monitor the patients, as we will see in Chapter 5. Moreover, the alerts produced by the agents can be transferred in an interoperable way as we saw in Chapter 3.

4.2 Architecture of MAGPIE

The main idea behind the MAGPIE agent platform is that the patient environment in a PHS and the abstraction of agent environment in Multi-Agent Systems (MAS) can be linked together. Thus, when such link takes place the patient environment of a PHS becomes a source of health-related information that can be exploited by the agents of a MAS. In that scenario, the goal for the agents is to keep track of the status of the patient, and perform an action when a potentially dangerous situation is detected. Moreover, in MAGPIE these monitoring agents are embedded in Android based mobile devices, so that the computations of the monitoring task are close to the patient and distributed over the Tier 2 of the particular PHS being deployed. This strategy of distributing the computations, rather than performing them in a centralized server, is intended to improve the scalability of PHS, which has been reported to be a primary requirement in that kind of systems [103]. This requirement is specially important if we take into account that human population is getting older so that in the future there will be a major prevalence of chronic diseases like for example Diabetes Mellitus (DM). For that particular disease, it is expected that its prevalence will increase 54% in 2030 compared with the worldwide population affected by this disease in 2010 [123].

To integrate in a framework the use of agents in Android, the architecture of MAGPIE is divided in two main levels (Figure 4.1). The uppermost level is the Conceptual Level, which models concepts from MAS like the agent environment, the agents, and all the interactions taking place between these components. The second level is the Android Integration Level, which corresponds to all the classes, interfaces, methods, etc. that act as an adapter between the Conceptual Level and the Android operating system. There is also a third layer integrating the use of Bluetooth sensors in the framework. These three layers detailed in the next subsections.

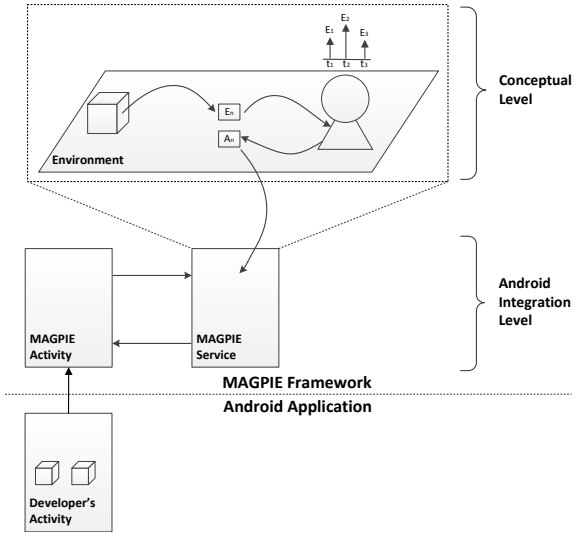


Figure 4.1: Architecture of the MAGPIE agent platform

4.2.1 Conceptual Level

The Conceptual Level of MAGPIE consists of three different main components that can be mapped to the elements of a publish/subscribe system [53]. These three components are: i) *agents*, as subscriber entities to the events happening into the environment, which are responsible to monitor the status of the patient and produce actions according to that status; ii) *context entities*, which are abstractions that encapsulate a source of information from the real world, and publish into the environment events related with the patient like a physiological measurement; and the iii) *environment*, which is an entity that acts as an event service, by mediating the interactions taking place between the context entities and the agents.

As stated before, in MAGPIE events flow through the environment from context entities to agents. To do this process in a handy way, agents express to the environment in which services provided by the context entities are they interested. In this way, the environment can distribute the events, by checking its type, to only those agents that are interested on them. Once an agent perceives an event, the agent processes the event and it produces an action that can be propagated to the User Interface (UI), so that at the application level the developer can treat it. This design strategy, following a publish/subscrbe pattern, where a central component is responsible for distributing the events shields the agents and context entities form knowing the implementation details about each other.

A key aspect of the environment is that it has a life cycle (Figure 4.2) that runs continuously on its own thread of control, different from the UI thread. For not draining the device's battery unnecessarily, the environment life cycle is implemented following the guarded suspension pattern [30]. The use of this pattern in our monitoring scenario involves that the environment pauses its execution until a certain precondition is satisfied. This condition that needs to be satisfied to awake the environment life cycle is the reception of a new event, or in other terms, that the environment's queue of events is not empty. When this condition happens, the environment continues its life cycle by dispatching the last event from the queue to the corresponding agents, and eventually notifying the possible actions produced by the agents before suspending its execution again.

4.2.2 Android Integration Level

The Android Integration Level provides the functionality to integrate the Conceptual Level with the Android operating system. This integration is done by means of coupling two main Android components: Activities and Services. An `Activity` is a class that represents a graphical interface displayed in the device's screen to interact with the user, and a `Service` is a class that runs in the background to perform long-running operations, which supports interaction with remote processes. In MAGPIE specific implementations of this components, named `MagpieActivity` and `MagpieService`, work together in a transparent way for the application developer.

The purpose of a `MagpieActivity` is twofold. First, it binds to the `MagpieService` to pro-

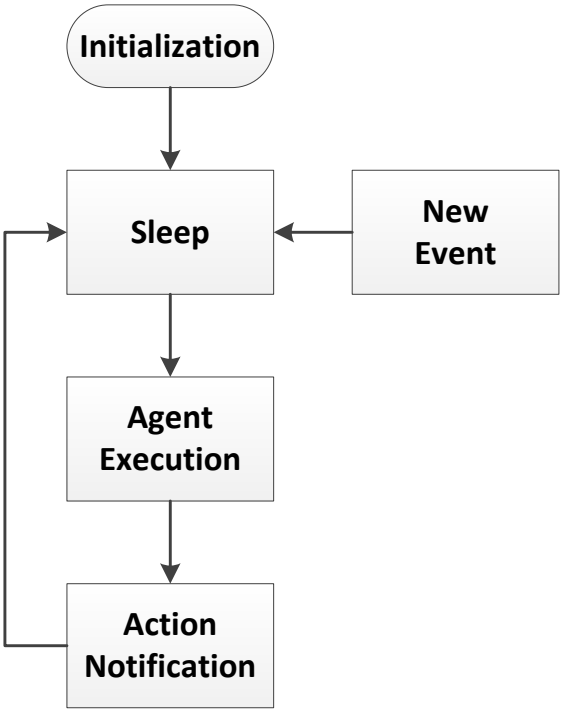


Figure 4.2: Environment life cycle

vide a communication channel with the `MagpieService`. In other words, it links the application UI with the MAGPIE Conceptual Level. Second, it provides inversion of control by defining hook methods that are are integrated in the `Activity` and `Service` life cycle, so that they are automatically called by Android. By implementing these methods the application developer provides her specific logic to the application in relation with the agent platform. In particular these methods are `onEnvironmentConnected()` and `onAlertProduced()`, which are defined in the interface `MagpieConnection` (Figure 4.3). The former is called when the `MagpieActivity` has finished the binding process with the `MagpieService`, and its purpose is to instantiate and register the agents into the environment. The latter is called when an agent produces an alert, with the aim of propagating it back to the UI. Thus, the developer can process it in several ways like displaying the alert to the user, storing it in a local database, or sending it to a remote server, etc.

```
public interface MagpieConnection {
  public void onEnvironmentConnected();
  public void onAlertProduced(LogicTupleEvent alert);
}
```

Figure 4.3: Hook methods defined in the MagpieConnection interface

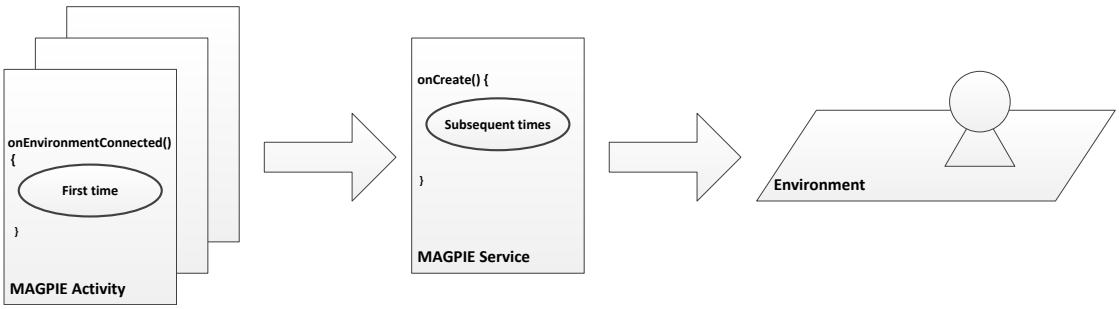


Figure 4.4: Pattern to register agents into the environment

The `MagpieService` is a bound service, which means that it offers a client-server interface that allows other components to interact with it. As a bound service it only runs as long as another application component, like a `MagpieActivity` in our case, is bound to it. In this communication act, the `MagpieService` plays the role of a server and its main tasks are to start, run and persist the state of the MAGPIE Conceptual Level. Specifically, to persist the state involves to take a snapshot of the agents running in the platform when no client is bound to the service, i.e. when the application is destroyed, and to recreate them when needed, i.e. when the application is restarted. As stated before, agents are registered into the environment from a `MagpieActivity`, but this only takes place the first time that the application is running. The subsequent times the agents are recreated and registered directly form the `MagpieService` (Figure 4.4). Thus, the state of the agents is not affected by the life cycle of Android components.

4.2.3 Sensor Communication

The Conceptual Level and the Android Integration Level offer functionalities that can be exploited in a PHS, like monitoring physiological values and provide feedback on the status of the patient by analyzing these values. However, by using just these two layers the physiological values must be provided manually by the patient via the UI of application running on the Android device. To fully support PHSs these functionalities have been extended to integrate Bluetooth sensors from a BAN, so that the physiological values can be also reported automatically in a more pervasive way. The MAGPIE agent platform provides support to generic Bluetooth sensors, and in particular supports the Zephyr's BioHarness [147].

BioHarness is a monitoring module that incorporates different sensors to monitor parameters like the heart rate, the R-R interval, the breathing rate, the posture, etc. A complete list of all data measured by this sensor can be found on its data sheet [21]. The sensed values can be recorded on the device or transmitted wirelessly by Bluetooth. Moreover the device can be attached to the body by means of a chest strap, or a shirt specially designed to hold it. All these characteristics makes it an ideal sensor to be used in PHSs.

Figure 4.5 shows the minimum set of components/entities involved in a monitoring appli-

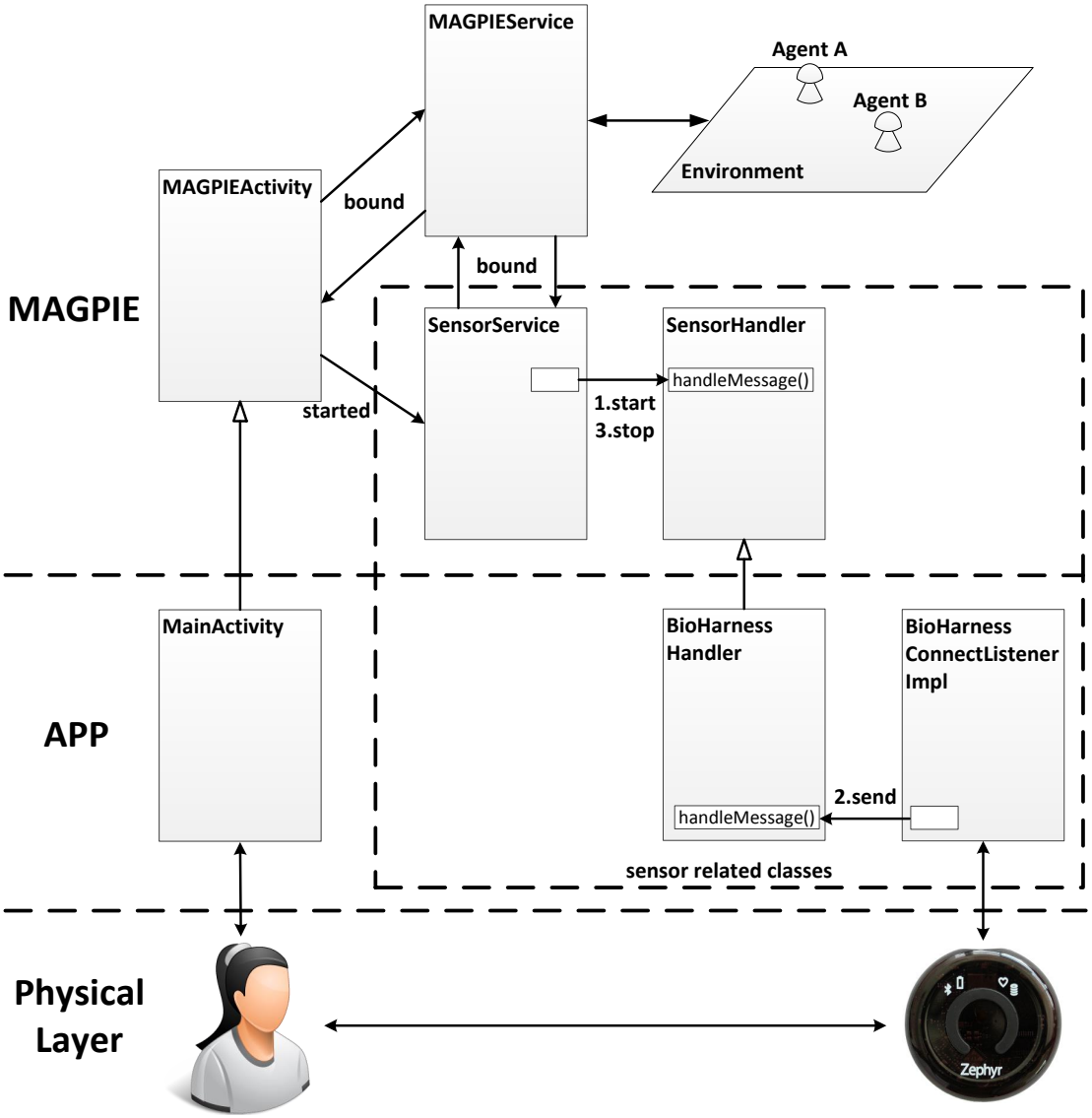


Figure 4.5: Architecture of the MAGPIE agent platform with the integration of BioHarness

cation working with the sensor, and how they are related to each other. From the developer's perspective these components conform three different levels. First, the i) *Physical Layer* concerns the user of the PHS, i.e. the patient, and the hardware of the sensor being used. Second, the ii) *Application* level is related to all the Java classes that the developer must program for the Tier 2 of the monitoring application. Third, the iii) *MAGPIE* level corresponds to the classes that form the core functionality of the agent platform. Notice that for the shake of clarity only the more important components of the MAGPIE architecture are shown in the figure.

Some of the MAGPIE components introduced in the previous sections provide extra funtion-

ality to handle the communications with the sensor. The Java classes from the MAGPIE level involved in the communications with the sensor are the following:

- `MAGPIEActivity`: provides the methods to start and stop the connection with the sensor, which are respectivey `connnectToSensor()` and `disconnectSensor()`. These two actions would be typically associated to a UI event like pressing a button. Additionally, the method `sensorConnectionResult()` is called at the end of the connection attempt to notify the user the outcome of the connection. This class must be extended in the *Application* level to provide the specific logic of these methods.

- `MAGPIEService` and `Environment`: the fact of having a sensor is transparent to these two classes, and therefore they do not provide extra functionality.

- `SensorService`: is an Android `Service`, launched by the `MagpieActivity` when the user starts a new Bluetooth connection with the sensor. As it is expected that the sensor will send measurements continuously, e.g. every second in the case of the BioHarness, this service launches its own thread of control to handle the communications with the sensor outside the UI thread. Thus, non blocking the UI at each new measurement provided by the sensor. The `SensorService` also binds to the `MagpieService` to route to the environment the sensor's measurements.

- `SensorHandler`: is an abstract Android `Handler` that implements the functionality to communicate with a generic sensor. This class manages the connections between the `SensorService` and the sensor itself by using Android messages. The `SensorHandler` class must be extended by the developer to provide the specific way of starting and stoping the sensor, and to handle the events that will be finally transferred to the environment.

The classes from the *Application* level provide the specific functionality to the PHS being deployed. These classes are the following:

- `MainActivity`: this class in an `Activity`, as it extends the `MagpieActivity` class. The `MainActivity` is the application's entry point to situate the agents on the environment, send events from the UI to the environment, start and stop the communications with the sensor, in addition to provide the specific logic to handle the application's UI.

- `BioHarnessHandler`: this class is a `Handler`, as it extends the `SensorHandler` class. The `SensorHandler` implements the methods from the interface `SensorConnection` (Figure 4.6). In these methods the developer specifies i) how to connect with the BioHarness, ii) how to stop the connection with it, iii) how the messages coming from the BioHarness must be processed before sending them as events to the environment, and iv) how to process the alerts coming from the environment.

```
public interface SensorConnection {
  public int onStartConnection();
  public void onStopConnection();
  public MagpieEvent processSensorMessage(Message message);
  public void onAlertProduced(LogicTupleEvent alert);
}
```

Figure 4.6: Hook methods defined in the `SensorConnection` interface

- `BioHarnessConnectListenerImpl`: this class extends the `ConnectListenerImpl` class from the BioHarness Android API. This class specifies which of the packets provided by the BioHarness sensor must be processed, and how to do it before sending the information to the `BioHarnessHandler`. The selection of the packets is based on their identification codes that are shown in Table 4.1. Each packet contains a set of information that can be queried and manipulated. As example, the General Data Packet contains information about the heart rate, the breathing rate, the posture, etc. A complete relation of the data contained in each packet is detailed in the data sheet of the sensor [21].

Table 4.1: Identifiers for BioHarness packets

Message type	ID
Accelerometer Data	42
Breathing Data	33
ECG	32
Event Data	44
General Data	32
Logging	75
Summary Data	43
Heart Rate R-R	36

4.3 MAGPIE Agents

The agents in MAGPIE are cognitive computing entities deployed in the agent environment, which have reasoning abilities like planning, decision making, and temporal reasoning. Their main goal is to assist the patient and the doctor using the PHS by carrying on the task of monitoring the patient, and providing feedback when an abnormal situation is detected. The agents in MAGPIE share a similar architecture with agents from other platforms like PROSOCS [125] and GOLEM [29]. As in these two platforms agents are composed by two main parts: a body and a mind. The agent body is the component responsible for situating the agent mind in the en-

vironment; while the agent mind is the component responsible to produce the actions according to the perceived events and how the medical knowledge is modeled in the particular mind.

The agent body is composed by sensors for the agent to be able to perceive events happening in the environment, and effectors to produce actions in the environment. Internally, the body has a queue where the perceived events are stored. When a new perception occurs, the agent body starts its own life cycle that consists on communicating the new event to the mind, and ask the mind to perform an action. If the event just notified to the mind produces an action, then the action is notified back to the body which can perform it on the environment, so that the environment can finally notify the action to the *application level* for its processing.

The agent mind is the cognitive part of the agent. For the purpose of a monitoring application the agent mind is modeled as a reactive mind [118], which perceives events from the environment through the agent body, i.e. physiological measurements from the patient; has a set of behaviors, i.e. the medical knowledge modeled in the mind; and produces actions, i.e. alerts according to its medical knowledge and the events perceived. To achieve their goals, two different agent minds are defined: a declarative mind based on Prolog, and an imperative mind based on Java. These two types of mind are discussed next.

4.3.1 Prolog Mind

The main characteristic of the Prolog mind is that it is based on a Prolog engine integrated with Java [47], so that it can be used in Android applications. Prolog is a logic programming language used to solve problems involving *objects* and their *relationships*. As example, the sentence "John has fever" declares the fact that there is the "having" relationship between the objects "John" and "fever". A Prolog program can also define rules such as "Someone has fever if his/her temperature is above 37.5°C". The Prolog mind is conceived for monitoring more complex rules than the one given as example. In particular, the medical knowledge in the Prolog mind models temporal patterns of physiological parameters like the ones described in Section 3.3.4 from the previous chapter. The formalization of these temporal patterns is based on the Event Calculus (EC) [89], which is introduced in Appendix C. Finally, how EC is used in the medical scope of this thesis is explained in details in Chapter 5.

The Prolog mind has an internal cycle that models the different steps involved in the monitoring task. This cycle follows a reactive agent pattern, which is specified as follows:

$$\begin{aligned} \text{agent\_cycle}(T) \leftarrow & \\ & \text{perceive}(P,T), \\ & \text{act}(A,T), \\ & \text{update}(A,T), \\ & \text{now}(T_{new}), \\ & \text{agent\_cycle}(T_{new}). \end{aligned} \tag{4.1}$$

Each Prolog mind cycle can be decomposed in five different predicates. Predicates in the text are referenced as predicate/N, where predicate is the name of the predicate and N the arity of the predicate, i.e. its number of arguments. In the first step the agent mind perceives events P coming from the environment, revises its knowledge base, decides for an action A to be performed at time T, and produces the action in the agent environment by pushing it to the agent body. In the next step the agent updates the knowledge base with the knowledge of having performed the action and starts a new cycle at time T_{new}. In more details, the perceive/2 predicate simply asserts events in the agent mind to modify the model that the agent has about the patient. The act/2 predicate simply checks for alerts in the agent mind, and if these hold an action is produced. The update/2 predicate is similar to the perceive/2 predicate, but it rather asserts internal events, such as events linked to actions performed in the agent environment.

4.3.2 Java Mind

The Java mind is able to run behaviors programmed in Java. In the context of the Java mind a behavior is defined as a task that can be carried out by an agent in response to an event happening in the environment. A Java mind can run N different behaviors defined by the `Behavior` class, and a particular behavior can be used by different Java minds, which makes this approach very modular. Agents with a Java mind can complement the tasks performed by agents running a Prolog mind, like for example computing statistics on the events happening in the environment.

The `Behavior` class defines a set of optional fields that are i) the *agent* that allows method invocation from the body or the mind, ii) the Android `Context` class that allows the agent to interact with the UI, and iii) a *priority* number that can be set to the behavior. A specific behavior is implemented by extending the `Behavior` class and implementing the two methods from the `IBehavior` interface (Figure 4.7). These two methods specify when the behavior must be triggered and what is the task to be performed by the behavior.

```
public interface IBehavior {
  public void action(MagpieEvent event); // What?
  public boolean isTriggered(MagpieEvent event); // When?
}
```

Figure 4.7: Methods of the interface `IBehavior` that define the operations of a behavior

The Java mind has a life cycle to run its behaviors that is composed by three steps. First, the events in the environment are perceived by the mind through the body. Second, the mind plans which are the behaviors to be executed. Last, the mind produces actions by executing its behaviors. The Java mind has two different strategies for executing the behaviors: i) a *sequential* strategy, where behaviors are executed in the same order that where defined; and a ii) *priority* strategy, where the behaviors are executed according to its priority. Different strategies can also be defined by implementing the abstract method `executeBehaviors()` of the Java mind.

4.4 Interaction Patient - System

A complete PHS developed with MAGPIE involves at least the patient as a user, but there are also other technological components like agents, the agent environment, sensors, and web services interacting to each other. The interactions taken place between the patient and the different components of the system are depicted in Figure 4.8. In particular, the figure shows the case where an event produces an alert relevant for the doctor and the patient. For the shake of clarity the figure shows only the interactions when having one sensor and one agent, but a patient can have multiple instances of these components.

In the first place the patient activates the Android application, and turns on the sensor(s) used for the continuous monitoring. At this point the agent environment starts the execution of its life cycle by waiting for the reception of physiological measurements produced by the sensors. The environment approach of waiting for events rather than being continuously running is intended to extend the battery of the device as much as possible. Once the sensor measures a physiological value, the context entity associated with the sensor encapsulates and forwards it to the environment's queue of events. The environment then activates all the agents interested in that particular event acting thereby as the mediator in the publish/subscribe pattern. The agent body is responsible for perceiving this event from the environment and send it to its agent mind to evaluate if an alert has to be triggered. In the next step the agent mind activates its internal cycle. First, it updates its internal state with the event just perceived. Second, it tries to achieve its goals by revising the monitoring rules defined for the patient. Finally, in the case that the event triggers a particular rule, the agent creates an action representing the alert to be notified. This action is performed into the environment through the agent body and notified to the patient through the UI of the mobile application. The environment also forwards the alert to a context entity that is responsible to redirect it to a web service located a the hospital, so that it can be added to the electronic health records of the patient and notified to the doctor.

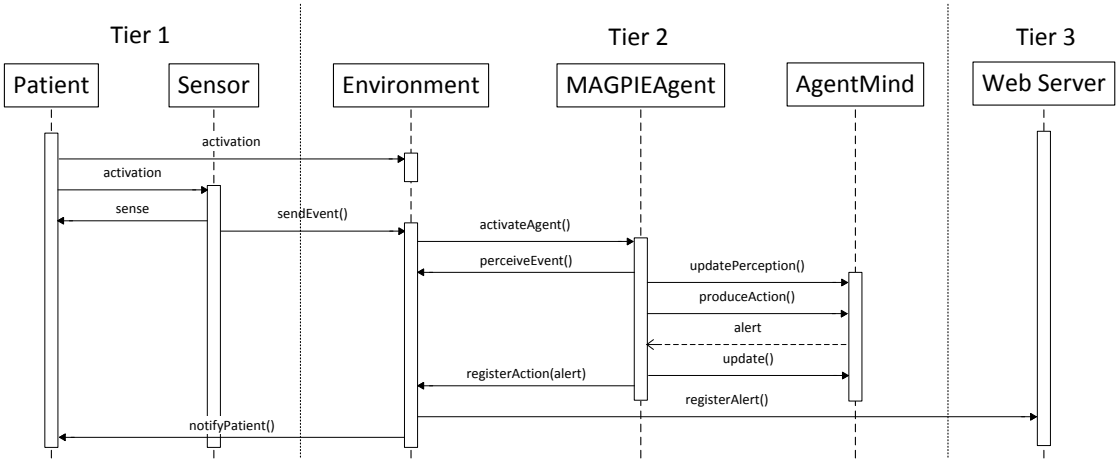


Figure 4.8: Interactions taken place between the tiers of the system for monitoring the patient

4.5 Evaluation

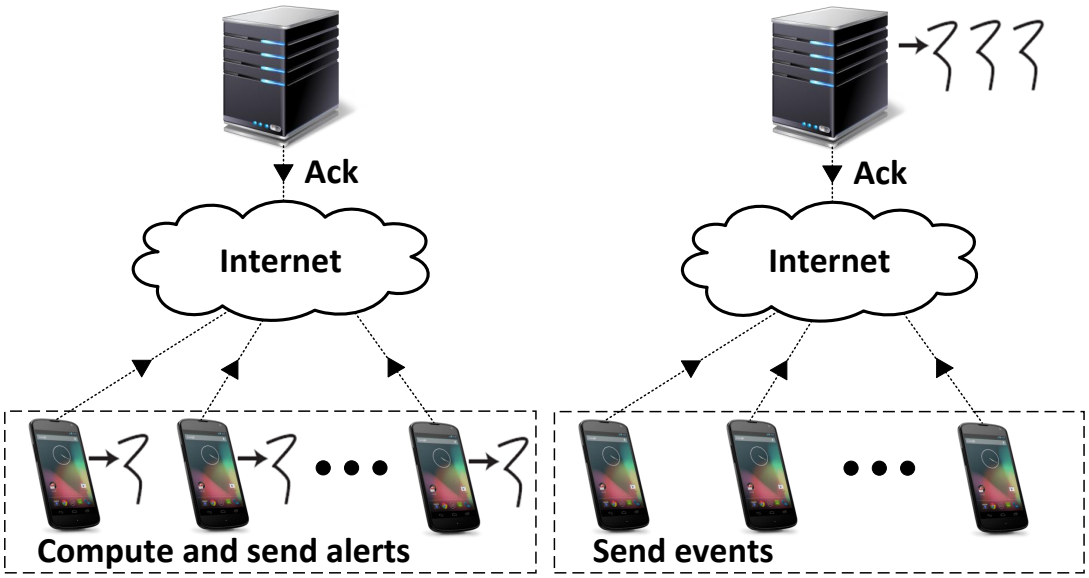
The typical approach to analyze the patient's monitoring data in PHSs is centralized, as it is all analyzed in the Tier 3 [133]. In contrast, in a PHS deployed with MAGPIE the computation to trigger the alerts is done in the Tier 2, so that it is distributed among the patients' mobile devices. Thus, the Tier 3 is free from doing this task, although the alerts must be sent to Tier 3 to notify them to the doctor.

To measure the scalability of both approaches, a simulation has been realized using the Amazon Web Services [5]. To compare the performance of the two approaches, a series of Amazon EC2 instances of the same type t2.micro have been used. The Amazon EC2 instances are virtual servers running in the cloud, which are easy to set up. The t2.micro instance comes with an Intel Xeon Processor running at 2.5 GHz, and 1 GB of memory. In total, eleven t2.micro instances have been used for the simulation. Ten instances for running the clients representing the patient's mobile devices in Tier 2 and one for running the monitoring server in Tier 3. Thus, in terms of hardware the clients and the server are always identical for both approaches. This is an important fact in order to compare the two strategies with the same conditions.

The simulation consists on distributing a certain number of clients over the ten EC2 instances, which run simultaneously performing the same operations (Figure 4.9). In particular, for the simulation of the MAGPIE approach, each client runs its own agent, which it is forced to trigger an alert composed by two events and send it to the remote monitoring server. The server has been developed using the Spring Framework, and its task is to store the alert in a repository and to send back an acknowledge message to the client. In the centralized approach, despite each client having its own agent, all the agents run in the server. In this approach, each client sends two events that force its agent to trigger an alert that is first stored in a repository and then notified back to the client as an acknowledgement message. In both cases the latency is measured as the elapsed time from the generation of the first event until the client is notified with the acknowledgment. The two simulated scenarios described above are partially distributed in the sense that each EC2 instance holds a certain number of clients, which inherently can interfere to each other. Thus, a real fully distributed scenario with one client per machine is expected to perform better. This is specially remarkable for the MAGPIE approach where the clients do a more intensive work than in the centralized approach.

Although there are other existing agent platforms for Android, which have been analyzed in Chapter 2, it is important to point out that a performance comparison between them in the medical scope of this thesis is not possible. In particular, not all the platforms have the source code available, and all of platforms have not been designed to run the expert system subject of this thesis. Here the evaluation focuses on how the expert system behaves when considering two different architectural approaches.

Figure 4.10 shows the results of the simulation, which are computed over 100 repetitions for each experiment. The results suggest that computation capabilities of smartphones should be used in PHSs. While the MAGPIE approach the latency has a flat response at around 50 ms from



(a) MAGPIE approach: computations distributed between the clients of the Tier 2

(b) Centralized approach: computations realized in the server located in the Tier 3

Figure 4.9: The two different scenarios used for evaluating the scalability

50 to 200 simultaneous patients, the centralized approach increases linearly with the number of patients. This difference is due to the fact that in the centralized approach, the threads launched by the server to process the client's requests must do a more intensive work in comparison with the MAGPIE approach, so that the interaction between the threads is bigger. Although the latencies obtained are not critical in a real scenario, the increasing tendency of latency in the centralized approach makes it unpractical for handling a big data scenario with a huge number of patients being monitored at the same time. The centralized approach presents also a higher variability compared with the MAGPIE approach. This difference is influenced by how the threads are scheduled by the server to process the client's requests.

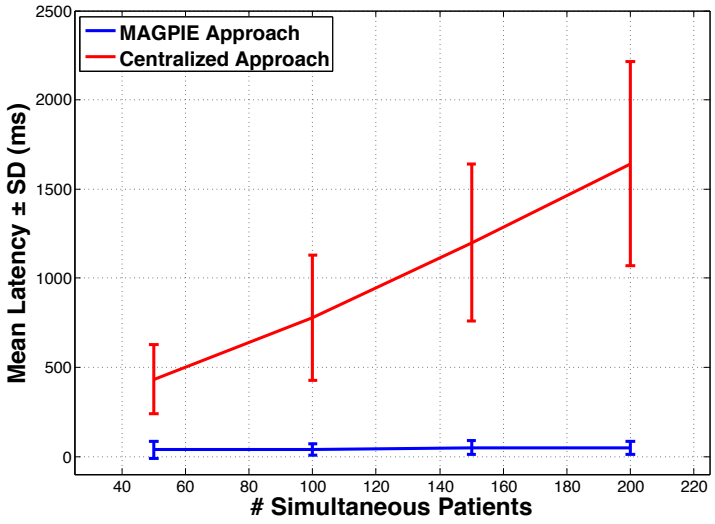


Figure 4.10: Simulation results for latency in the MAGPIE approach and in the centralized approach when increasing the number of patients using the system. The experiment consists on triggering simultaneously an alert for each patient. The alert is composed by two events for both approaches. The mean and standard deviation are calculated over 100 repetitions

Chapter 5

Formalization: Graphical Rules for Monitoring Chronic Diseases

Contents

Up to now our Personal Health System (PHS) provides interoperability through the use of healthcare standards, and scalability with the MAGPIE agent platform, as we have seen respectively in Chapter 3 and Chapter 4. The agents in MAGPIE provide the intelligence to the system by monitoring the patient by means of monitoring rules and providing feedback in terms of alerts. However, such intelligence is static in the sense that in order to change it, it must be reprogrammed and fed into the system. The systems' users able to provide the medical knowledge to monitor the patient are the medical doctors, but as it is not expected that a medical doctor has the knowledge on the logic programming needed for doing that task, this chapter introduces how the formalization of such monitoring rules can be done in a graphical way. Such graphical-based formalization is done through a web application whose concepts are introduced in Section 5.1. This section also details the integration of this web application with MAGPIE. Section 5.2 explains how the graphical representation is related to the underlying logic rules. Section 5.3 specifies different temporal patterns for Diabetes Mellitus (DM) that can be specified with the proposed application. Finally, Section 5.4 evaluates the ability of the method in detecting such temporal patterns in real data taken from patients suffering DM Type II.

5.1 Web Application for Monitoring Rules

The agents of the MAGPIE agent platform, introduced in the previous chapter, must provide alerts according to a set of monitoring rules. Medical doctors are the users of the PHS who have the knowledge to define such monitoring rules. However, they may not have knowledge on logic programming to define specific monitoring rules that are understandable by the agents. To help medical doctors to play their role in an independent way, one of the components of our PHS is a web application where doctors can program monitoring rules for the agents in a graphical way.

In the context of this thesis, a monitoring rule is defined as a combination of events that trigger an alert to be notified to a medical doctor; where an event is considered as the measurement of a physiological parameter, like the glycemia, the blood pressure or the weight, categorized as high, normal or low. By combining different events two kinds of monitoring rules can be defined:

- **Complex rules:** involve the combination of two or more events in a given time window, where it is not considered the order in which the events happen.
- **Sequential rules:** involve the sequence of two or more events in a given time window, where the particular order in which the events occur matters.

Moreover, to provide personalized healthcare services, the medical doctor can define the high and low thresholds for each physiological parameter and patient, and therefore the normal range too. This is an important feature as a person can be considered to have the blood pressure high if it is 10/90 mmHg, but this measurement can indicate an improvement if the person had higher values in the past.

The creation of a monitoring rule follows the approach of a visual programming language, where different graphical elements are combined together to define the logic of a computer program. As stated before, monitoring rules are a combination of events, and therefore this is the only kind of graphical element that must be modeled, which in turn minimizes the complexity for creating the rules. Figure 5.1 shows the graphical design of an event, where *Event* defines the category and the name of a physiological parameter (i.e. high blood pressure); T is the time window given to the event, that is the amount of time after the specified event happens; and N is the number of times that the event must repeat for the given time window.

To create a monitoring rule the doctor must first select the type of rule, and then he can drag and drop the graphical elements representing the events that must happen in order to trigger the alert. The events are matched together vertically for complex rules and horizontally for sequential rules. Last, he can configure the parameters defining the event, and the message that must be displayed if the alert is triggered.

Figure 5.2 shows the relation between the graphical representation and the temporal representation of different rules that can be build with the web interface. Figure 5.2a shows the simplest rule that can be build, which is given as an illustrative example. This rule states that an

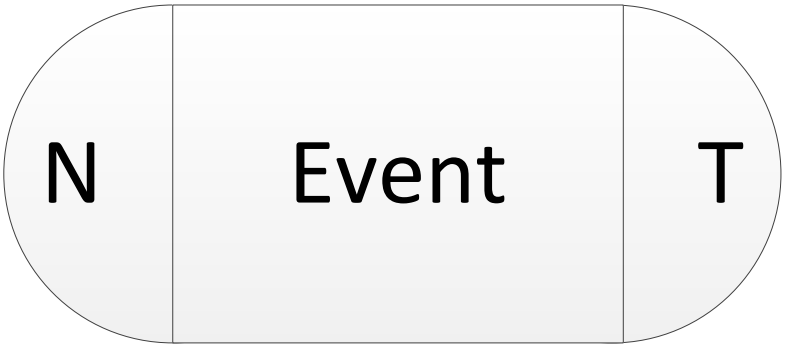
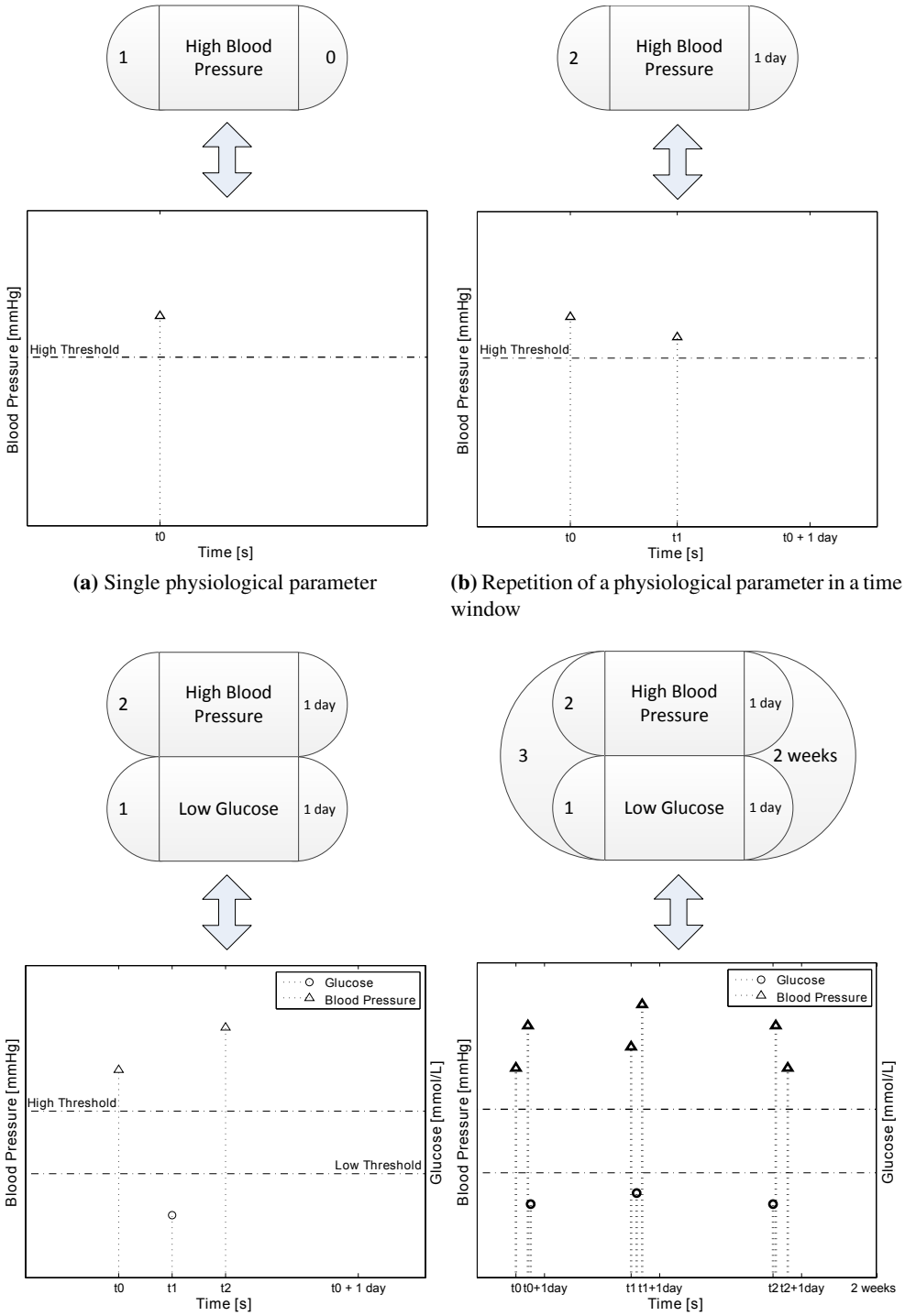


Figure 5.1: Graphical representation of an event, where *N* is the number of repetitions and *T* the time window for the event

alert is triggered if there is a single event of high blood pressure at time t_0. However, the web interface is intended to build complex and sequential monitoring rules involving several events. Figure 5.2b shows an example of a monitoring rule being complex and sequential at the same time, as it involves just one physiological parameter with a particular category. This rule states that an alert is triggered if the patient has two measurements of high blood pressure in a time period of one day. Figure 5.2c shows an example of a complex rule involving three events with two physiological parameters. This rule states that an alert is triggered when there are two events of high blood pressure and one event of low glucose in a time period of one day, where the time begins counting when the first of the events happens. The web interface also allows the user to create meta-rules. As shown in Figure 5.2d a meta-rule is a rule composed with a rule. In this case an alert is triggered if the pattern of events defined by the previous rule repeats three times in a period of two weeks.

5.1.1 Integration with MAGPIE

In order that the MAGPIE agents can use the rules defined through the web application, they must be sent to the patient's device running the agent platform. Figure shows the interactions between the elements involved in this process. In Tier 3 when new monitoring rules are defined by the doctor, these are stored in a local repository and are ready to be downloaded by the monitoring application through a web server. In Tier 2, a rest client context entity is responsible for connecting with this web server and getting these new monitoring rules. The rules are then encapsulated by the context entity as a rule set event and sent to the agent environment's queue of events for their processing. Once the environment is ready for processing events, it activates the agents interested in that particular rule set, which is loaded into the agent's mind after the previous rule set has been discarded.



(a) Single physiological parameter

(b) Repetition of a physiological parameter in a time window

(c) Combination of different physiological parameters in a time window

(d) Meta-rule combining the repetition of a pattern of events in a time window

Figure 5.2: Temporal and graphical representations of different monitoring rules

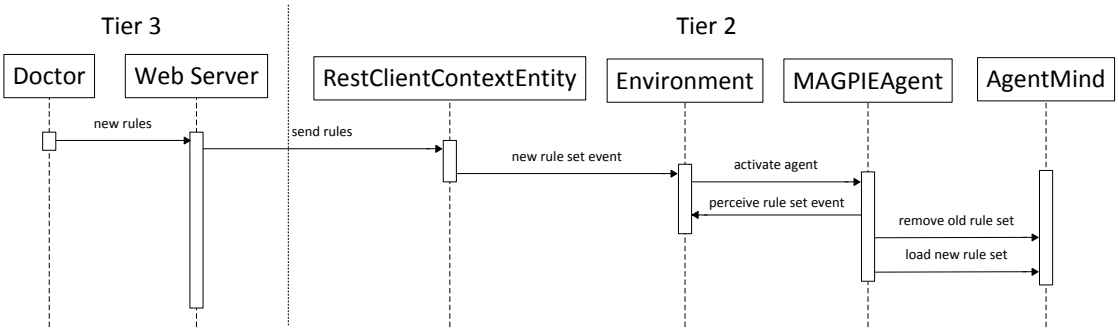


Figure 5.3: Interactions taken place for updating the monitoring rules of a patient

5.2 Knowledge Representation

To allow the agents to produce alerts according to the rules specified by the doctors, the graphical rules must be converted to a logic representation understandable by the agents. As discussed in the corresponding section of the previous chapter, the agents with a Prolog mind can deal with monitoring rules expressed as Prolog clauses. These Prolog monitoring rules produced for the agents are based on the Event Calculus (EC) [89] as the underlying formalism to deal with the temporal events happening in the agent environment. The EC is a formalism to represent actions and their effects, which is described in details in Appendix C. Figure 5.4 shows how the elements defined by an event are linked to its Prolog representation using EC. The monitoring rules are modeled using the initiatesAt/2 EC predicate, which means that an alert of a particular type is triggered at time T if the events defined in the rule's body happens. In the particular example of the Figure 5.4, an alert of type *high_blood_p* notifies a *message*, if at time T there is a blood pressure measurement whose systolic value is higher or equal to 160 mmHg and its diastolic value is higher or equal to 100 mmHg.

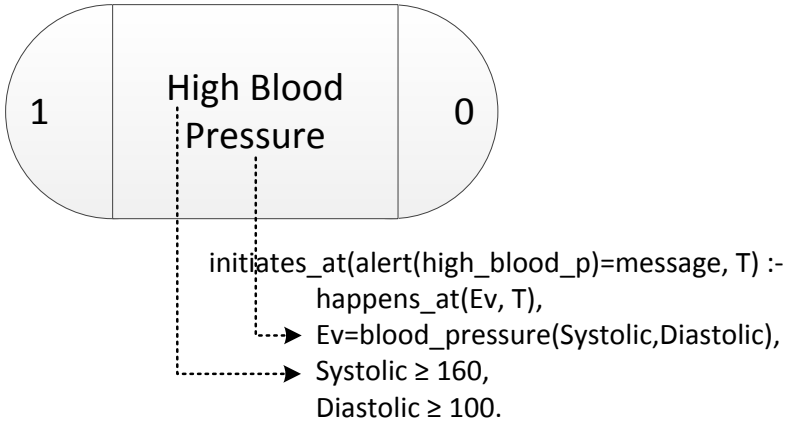


Figure 5.4: Relations between the graphical representation of an event and Prolog code in EC

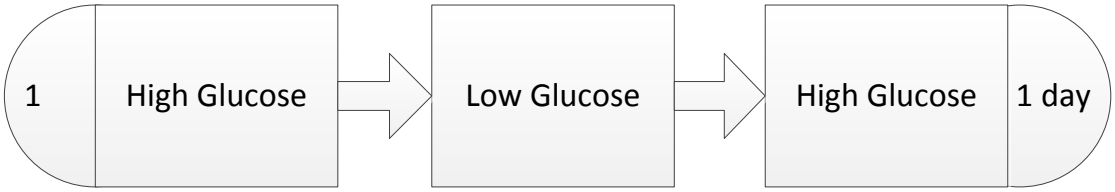


Figure 5.5: Sequential rule defining the glucose pattern: high → low → high

In the case of having a monitoring rule with more than one event, it can be derived from the previous example that the different events can be nested together taking into account the time in which they happen. Figure 5.5 shows an example of a sequential rule with these characteristics. This rule consists on three events defining the glucose pattern: high → low → high in a time period of one day, where a high glucose is considered a value above 8 mmol/L, and a low glucose a value below 3.8 mmol/L. The Prolog representation of this rule is as follows

$$
\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{glucose\_pattern}) = \text{message}, T) \leftarrow \\
&\quad \text{happensAt}(Ev1, Tev1), \\
&\quad \text{happensAt}(Ev2, Tev2), \\
&\quad \text{happensAt}(Ev3, Tev3), \\
&\quad Ev1 = \text{glucose}(Value1), \\
&\quad Ev2 = \text{glucose}(Value2), \\
&\quad Ev3 = \text{glucose}(Value3), \\
&\quad Value1 \geq 8, \\
&\quad Value2 \leq 3.8, \\
&\quad Value3 \geq 8, \\
&\quad Tev3 > Tev2, \\
&\quad Tev2 > Tev1, \\
&\quad \text{last\_day}(Tev1, Tev3), \\
&\quad \text{not happensAt}(\text{alert}(\text{glucose\_pattern}), Ta), \\
&\quad \text{last\_day}(Ta, T).
\end{aligned}
\tag{5.1}
$$

Three different time conditions must be satisfied in the rule (5.1)

- The event Ev3 must happen after the event Ev2, which in turn must happen after the event Ev1.
- The distance in time between the first and the last event must be less than one day.
- This particular glucose pattern must not have been triggered during the last day.

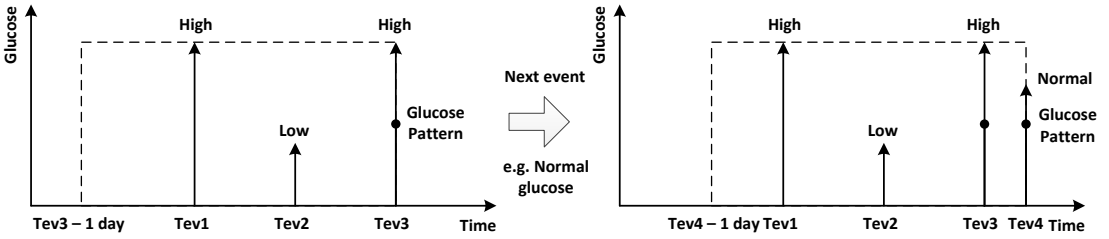


Figure 5.6: The inertia effect makes the rule trigger twice when it is not checked if the temporal pattern already happened in the temporal window

The second and third time conditions are checked through the last_day/2 predicate used in the rule (5.1) and defined in (5.2). This predicate computes if the distance in time between the two input arguments is less than one day. Notice that the absolute time is expressed in milliseconds, since Java's APIs use this unit for the epoch time. Similar Prolog predicates exist for the different time windows that can be specified in the web application.

$$\begin{aligned} &\text{last\_day}(Tev, Tfinal) \leftarrow \\ &\qquad Tinit \text{ is } Tfinal - 24*60*60*1000, \\ &\qquad Tev \leq Tfinal, \\ &\qquad Tev \geq Tinit. \end{aligned} \tag{5.2}$$

In the third time condition, this predicate is used in conjunction with the negation of the happensAt/2 EC predicate to check that the temporal pattern defined by the rule has not been triggered during the last day. This condition avoids alerting the doctor multiple times with events that already triggered an alert due to the common sense law of inertia [122]. As shown in Figure 5.6, inertia implies that everything remains on its state. In this case, the alert would be triggered for the second time by the same events, if the condition is not checked when a new event arrives.

The approach followed for defining sequential rules is not practical for defining complex rules. This is due to the fact that a graphical rule composed with many events will derive in a set of different initiatesAt/2 predicates, each one corresponding to a temporal permutation of all the events. To deal with this issue a more_or_equals_to/2 predicate has been defined. This predicate counts the number of facts in the knowledge base satisfying the conditions defined in its second argument, and returns true if it finds at least the same number of facts defined in its first argument. The definition of the predicate more_or_equals_to/2 is as follows

$$\begin{aligned} &\text{more\_or\_equals\_to}(Number, Expr) \leftarrow \\ &\qquad \text{findall}(\_, Expr, List), \\ &\qquad \text{length}(List, Val), \\ &\qquad Val \geq Number. \end{aligned} \tag{5.3}$$

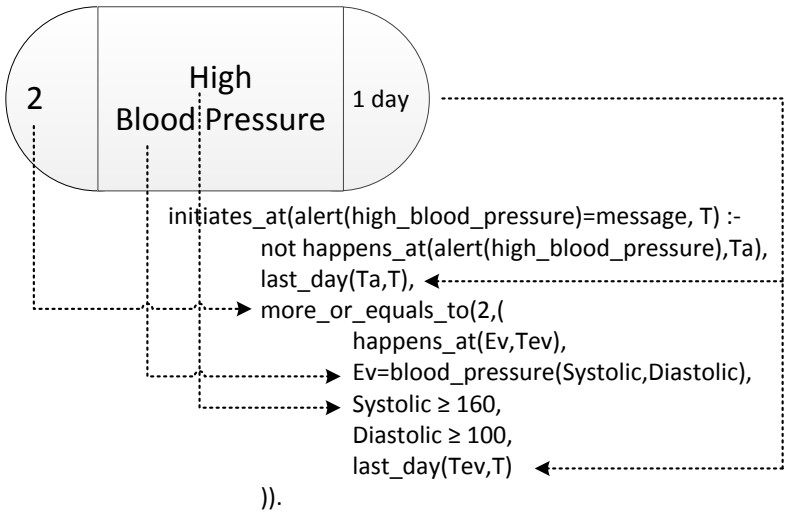
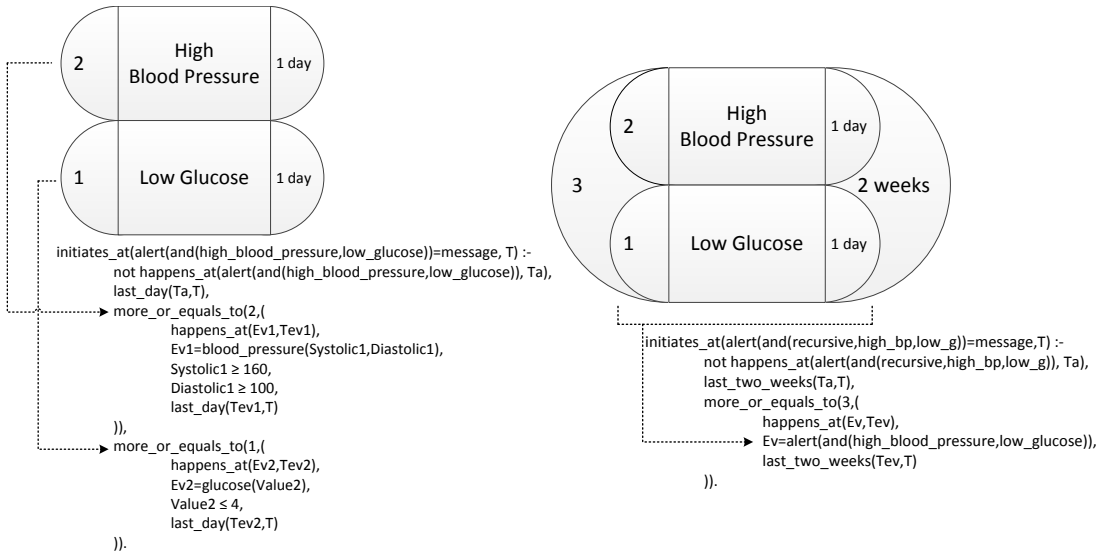


Figure 5.7: Use of the more_or_equals_to/2 counting predicate in a complex rule

As shown in Figure 5.7 the predicate more_or_equals_to/2 is used in the body of the complex rule to count the number of events satisfying the defined conditions. In the example, an alert is triggered if two conditions are satisfied. The first one is related with the predicate more_or_equals_to/2, which checks if in the last day there were at least two blood pressure events, whose systolic and diastolic values were higher or equal to 160 mmHg and 100 mmHg respectively. The second one, as in the sequential rules, checks that the same alert was not already triggered during the last day.

In the case of meta-rules, where the goal is to alert the repetition of a particular pattern of events in a given period of time, two different domain dependent EC predicates are created: the inner part and the outer part. The inner part, shown in Figure 5.8a represents a single occurrence of the pattern defined, and the predicate consists on the nesting of more_or_equals_to/2 predicates. The outer part (Figure 5.8b) consists on counting if the specified pattern of events happens at least a certain number of times in the given time window. The counting is done with the more_or_equals_to/2 predicate, where the event is the alert representing the inner part of the rule.

The complex and sequential rules discussed in this section follow a pattern that can be expressed in pseudocode. In summary, a complex rule is set to active at time T_e, if in a given time window [T_s, T_e] the number of events E_n satisfying the conditions C_n is at least equals to the expected number; and the alert has not been previously triggered within the time window (Figure 5.9). For a sequential rule, this is set to active at time T_e, if in a given time window [T_s, T_e] there is the sequence of the N events E_1, E_2, ..., E_N satisfying the conditions C_1, C_2, ..., C_N; and the alert has not been previously triggered within the time window (Figure 5.10).



(a) Nesting of events in a complex rule **(b)** Meta-rule composed with the complex rule

Figure 5.8: Examples of code produced for a complex rule and a meta-rule

```
IF
within a time window [Ts,Te] THERE IS
complex pattern NOT 'active' AND
# Events E_n Satisfying Condition C_n
THEN
complex pattern == 'active' at time Te
```

Figure 5.9: Pseudocode defining the structure of a complex rule

```
IF
within a time window [Ts,Te] THERE IS
sequence pattern NOT 'active' AND
Event E_1 at time T_1 Satisfying Condition C_1 ...
Event E_N at time T_N Satisfying Condition C_N AND
T_1 < ... < T_N
THEN
sequence pattern == 'active' at time Te
```

Figure 5.10: Pseudocode defining the structure of a sequential rule

5.3 Specific Diabetes Mellitus Rules

Observing blood glucose trends and patterns in diabetic patients has been reported to be beneficial [140], as it can help to address the cause of the problem. In this section we define three different patterns of interest that involve temporal evolution of glucose, blood pressure and weight. These temporal patterns are expressed in terms of complex and sequential rules, and are later used in the evaluation of the system.

- **Pattern 1:** Brittle diabetes, defined as a glucose rebound going from less than 3.8 mmol/L to more than 8.0 mmol/L in a period of six hours. This pattern can be expressed with a sequential rule as follows,

$$\begin{aligned}
&\text{initiatesAt}(alert(p1) = \text{'brittle diabetes'}, T) \leftarrow \\
&\quad \text{not happensAt}(alert(p1), Ta), \\
&\quad \text{last_six_hours}(Ta, T) \\
&\quad \text{happensAt}(Ev1, Tev1), \\
&\quad \text{happensAt}(Ev2, Tev2), \\
&\quad Ev1 = \text{glucose}(Value1), \\
&\quad Ev2 = \text{glucose}(Value2), \\
&\quad Value1 \leq 3.8, \\
&\quad Value2 \geq 8, \\
&\quad Tev2 > Tev1, \\
&\quad \text{last_six_hours}(Tev1, Tev2).
\end{aligned} \tag{5.4}$$

- **Pattern 2:** Pre-hypertension, defined as two events of high blood pressure in a period of one week. This pattern can be expressed with a complex rule as follows,

$$\begin{aligned}
&\text{initiatesAt}(alert(p2) = \text{'pre-hypertension'}, T) \leftarrow \\
&\quad \text{not happensAt}(alert(p2), Ta), \\
&\quad \text{last_week}(Ta, T) \\
&\quad \text{more_or_equals_to}(2, (\\
&\qquad \text{happensAt}(Ev1, Tev), \\
&\qquad Ev1 = \text{blood_pressure}(Sys, Dias), \\
&\qquad Sys \geq 130, \\
&\qquad Dias \geq 80, \\
&\qquad \text{last_week}(Tev, T))).
\end{aligned} \tag{5.5}$$

- **Pattern 3:** Gaining weight, defined respect the initial weight as going from a 2% lost to a 1% gain in a period of one week. The 2% and 1% values for this pattern must be set up by the doctor as the low and high thresholds respectively in the application. These thresholds are different for each patient, since each one has a different initial weight. For a particular patient having an initial weight of 111.5 kg this pattern can be expressed using a sequential rule as follows,

$$\begin{aligned}
&\text{initiatesAt}(alert(p3) = \text{'gaining weight'}, T) \leftarrow \\
&\quad \text{not happensAt}(alert(p3), Ta), \\
&\quad \text{last_week}(Ta, T) \\
&\quad \text{happensAt}(Ev1, Tev1), \\
&\quad \text{happensAt}(Ev2, Tev2), \\
&\quad Ev1 = \text{weight}(Value1), \\
&\quad Ev2 = \text{weight}(Value2), \\
&\quad Value1 \leq 109.3, \\
&\quad Value2 \geq 110.4, \\
&\quad Tev2 > Tev1, \\
&\quad \text{last_week}(Tev1, Tev2).
\end{aligned} \tag{5.6}$$

5.4 Evaluation

The evaluation of the proposed solution has been done using data collected with sensors from the COMMODITY$_{12}$ project. This project consists of a PHS for the monitoring of patients affected with DM. Three different sensors were used to collect physiological values from the patients. The sensors used are the following

- GlucoTel for capillary blood glucose measurements in mmol/L made six times per day on Mondays, before and after each meal; and one measurement during the morning, after breakfast, the rest of the days. This difference in the number of samples per day is because in the COMMODITY$_{12}$ project, the glucose values are correlated with Electrocardiogram (ECG) readings, and the capillary blood glucose test is an invasive technique for DM patients. These two facts make a compromise in the treatment. The accuracy of the blood glucose sensor is 0.22 mmol/L.

- PressureTel for the measurement of the blood pressure in mmHg twice a day; one measurement during the morning, after wake up, and one during the evening after dinner. The accuracy of the PressureTel sensor is 3 mmHg.

- WeightTel a scale used to measure the weight of the patient in kg once a day. The accuracy

of this sensor is 100g.

A total of 21 patients participated in the trial, which used the sensors described during a period of six weeks approximately. The patients that participated in the trials share common characteristics apart from being affected by DM Type II. They are between 50 and 70 years old. They are not very active persons, which influenced in the development of their DM. They are often obese, and can have also hypertension and dyslipidemia. Thus, they are at increasing risk of having cardiovascular issues.

For evaluating the ability of the proposed system in detecting temporal patterns in real data, a retrospective analysis on the data collected in COMMODITY$_{12}$ has been done. Table 5.1 summarizes the statistics of the dataset. The rows in samples per day indicate how well the patients follow the treatment described in this section. According to the treatment these values should be ideally 1.71 ± 1.89, 1 ± 0, 1 ± 0 for the glucose, blood pressure and weight respectively.

Table 5.2 shows the results concerning the triggering of the rules given the three previously presented selected patterns. The interesting aspect of these results is that given the patients data, the system is able to find the patterns defined by the doctors. In particular, the selected patterns where rather simple in the sense that they only combine one type of event, and based on the common practice of the medical doctors, so the system could detect the entirety of the selected patterns. This is significant because the ability to detect these patterns allow medical doctors to modify the treatment of a selected patient and thus the reaction time of the medical doctors is more effective.

Something remarkable about the presented PHS is that is meant to detect temporal patterns of physiological values that are human defined. This detection of patterns is based on the values reported by the measuring sensors, which means that the sensitivity and specificity of the system depends directly on the accuracy of the sensors used. Thus, the same system using different sensors could perform differently. However, the system can track when a sensor is reporting a physiological value that is far from its possible range, e.g. a 30 mmol/L glycemia, as specific rules for these cases can be created. Another thing to point out is on how the patients use the sensors. For instance, it might happen that a person other than the patient uses the scale or any other sensor, and the measured value is reported as if it was from the patient. In that case, this could lead to wrongly fire an alert. The previously mentioned issues are out of the scope of the system design and are subject for future research.

Table 5.1: Statistics describing the dataset composed of 21 DM Type II patients

Patient	Days	Samples per Day (Mean±SD)			Measurement (Mean±SD)			
		Glucose	Blood Pressure	Weight	Glucose (mmol/L)	Sys. BP (mmHg)	Dias. BP (mmHg)	Weight (kg)
1	42	1.71±1.96	2.33±1.14	0.95±0.21	7.45±1.64	128.06±8.27	77.04±4.93	108.71±1.43
2	44	1.69±1.88	1.86±0.42	0.98±0.15	10.53±1.72	143.78±9.68	79.63±6.50	84.77±0.98
3	42	1.95±2.23	1.93±0.41	1.00±0.31	8.24±1.76	126.06±7.41	73.42±4.86	94.95±0.81
4	42	1.57±1.81	1.90±0.37	0.97±0.15	9.31±2.27	128.49±7.59	66.98±6.40	62.67±0.39
5	42	2.00±2.26	2.36±0.66	1.07±0.26	6.85±1.64	150.63±9.28	89.04±8.70	106.34±0.57
6	41	1.66±1.87	1.83±0.44	1.00±0.00	9.49±2.40	143.52±12.48	80.95±7.92	93.70±0.43
7	44	1.93±2.15	3.39±2.17	1.05±0.37	7.31±1.69	126.52±8.10	81.10±5.42	78.50±0.85
8	34	1.65±1.69	1.82±0.39	1.03±0.17	7.06±1.34	133.56±8.34	69.68±7.51	83.51±0.52
9	31	1.84±1.95	1.97±0.55	0.97±0.18	5.68±1.06	139.87±12.91	79.70±5.98	85.47±0.56
10	14	1.86±2.03	1.93±0.27	1.00±0.00	9.86±2.73	126.93±7.19	68.96±3.39	89.63±0.48
11	32	1.88±1.70	1.91±0.39	1.16±0.77	6.21±1.28	134.90±12.67	81.05±9.10	67.73±0.51
12	42	1.60±1.60	1.98±0.41	0.98±0.15	9.27±1.99	120.54±7.78	68.43±5.69	86.80±0.39
13	42	1.71±1.84	1.90±0.48	0.90±0.30	10.21±3.36	136.83±10.44	81.16±6.82	77.51±0.61
14	30	1.73±1.64	1.97±0.56	0.97±0.41	6.92±1.02	121.20±6.63	67.85±4.77	99.16±0.43
15	42	1.69±1.73	1.83±0.58	0.98±0.15	6.39±1.40	90.77±6.85	61.29±5.69	58.12±0.68
16	42	1.67±1.80	1.98±0.41	1.05±0.22	7.52±1.49	127.92±6.90	79.08±5.45	98.39±1.22
17	40	1.55±1.71	1.70±0.52	1.00±0.45	11.35±2.41	127.49±7.50	82.07±5.16	84.47±0.81
18	36	1.92±1.80	2.00±0.93	1.00±0.00	10.32±2.95	156.28±11.68	92.25±7.62	116.61±0.83
19	41	1.73±1.83	2.00±0.39	1.00±0.00	6.05±0.96	124.74±8.25	70.41±5.12	95.64±1.21
20	38	1.84±1.90	1.76±0.63	1.00±0.00	6.67±1.12	120.34±8.12	54.61±7.16	68.59±1.05
21	32	1.34±1.12	1.97±0.59	1.13±0.61	9.10±2.48	137.81±9.21	79.83±5.26	95.33±0.46

Table 5.2: Detection of patterns in the dataset

Patient	Pattern 1	Pattern 2	Pattern 3
1	0	4	0
2	0	6	3
3	0	2	0
4	0	0	1
5	0	6	1
6	0	6	2
7	0	6	0
8	0	3	0
9	0	4	1
10	1	0	0
11	0	5	1
12	0	0	0
13	0	5	0
14	0	0	0
15	0	0	2
16	0	5	2
17	0	4	0
18	0	5	2
19	0	0	0
20	0	0	3
21	0	4	0

Chapter 6

Enhancements to Tier 3 of Personal Health Systems

Contents

Along the last three chapters we have developed a Personal Health System (PHS) that considers interoperability by generating Electronic Health Records (EHRs) (Chapter 3), scalability through the use of the MAGPIE agent platform (Chapter 4) and formalization of the monitoring rules used to monitor the patient (Chapter 5). In this chapter we introduce two different ways to share the information collected with this system in order to improve the outcomes obtained with this technology. Section 6.1 focuses on how data from a PHS can be shared between individuals. The approach consists on integrating the use of the MAGPIE agent platform into a Distributed Event-Based System (DEBS). Section 6.2 focuses on how data can be shared between institutions through a network protocol based on Multi-Agent Systems (MAS) called MOSAIC. This section gives emphasis on the security of this protocol.

6.1 Sharing Data Between Individuals: A DEBS Approach

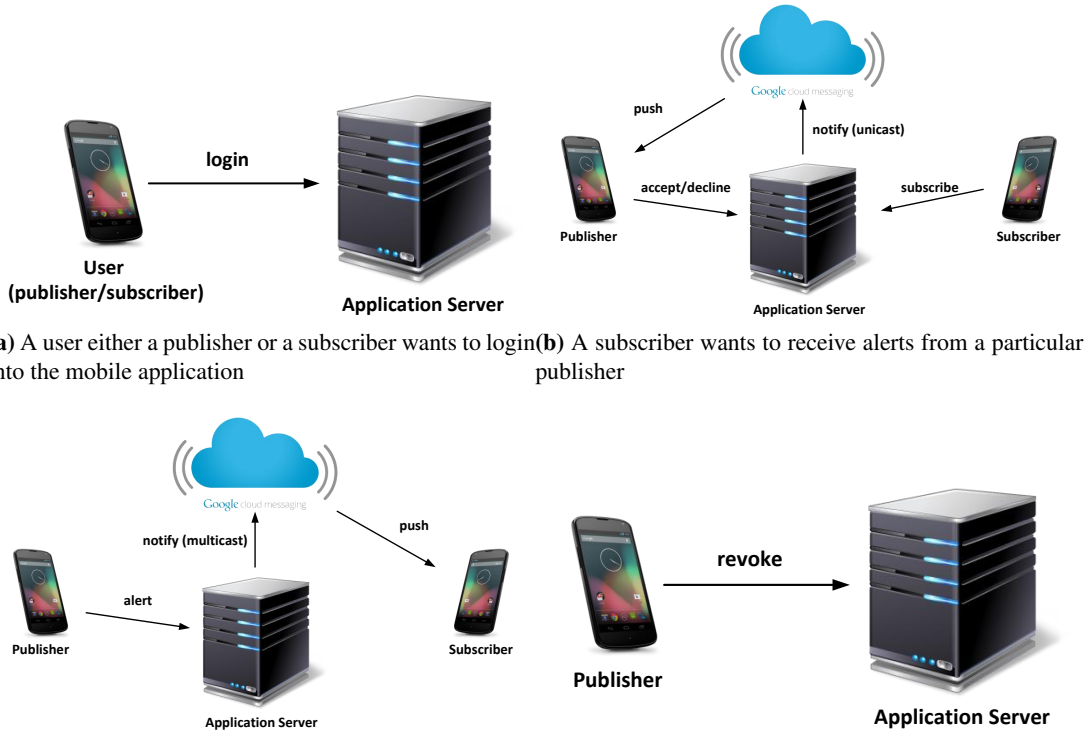
The MAGPIE agent platform, introduced in Chapter 4, and its integration with the BioHarness sensor focuses on the Tiers 1 and 2 of PHSs. In this section we cover the integration of MAGPIE in a DEBS manner, which covers the Tier 3 of PHSs. The idea behind this integration is that in addition to healthcare professionals, other peers and relatives can also be notified about the health-related events happening to the patient. This extends the roles of the patient to not only an active participant, but also an information sharer, and a peer leader of its own healthcare data. This shift in the role of the patient also involves that the role of the physician evolves too, which may become more a healthcare consultant and collaborator. This patient-driven delivery model of healthcare services has been reported as being beneficial for the patients [129], as it can help them to get a better understanding of their illness.

In a generic DEBS an *event* is defined as "*any happening of interest that can be observed from within a computer*" [102]. In our PHS scenario an event is an alert produced by the measurement of a physiological parameter, i.e. a patient suffering an hypoglucemia. The alerts can be rather simple as just taking one physiological parameter like the blood glucose, the blood pressure or the weight, or be more complex like the temporal patterns introduced in Chapter 5. In DEBS there are two actors in relation with the events: the producer of events named the publisher, and the consumer of events known as the subscriber. In our case the publisher corresponds to the patient, and the subscriber can be the doctor in charge of the patient, a relative, or another patient suffering the same disease.

We developed a prototype system that incorporates the above-mentioned concepts, which consists on three main components: i) an Android application that uses MAGPIE on its core to produce events, ii) an application server as event notification service, and iii) the Google Cloud Messaging (GCM) service for delivering push messages. We defined four main scenarios or interactions taking place between these components, which are depicted in Figure 6.1 and described below:

- **Scenario A:** a user, which can be a publisher or a subscriber wants to use the mobile application. Thus, the application requests the credentials to the user (i.e. username and password) that are validated by the application server. If this validation process succeeds the user is allowed to use the mobile application. This process is done using the OAuth 2.0 authorization framework [66].

- **Scenario B:** a subscriber wants to subscribe to the alerts produced by a particular publisher. To accomplish this process the subscriber must send a request to the application server specifying the username of the publisher. The application server redirects this request to the GCM service, which notifies about it to the publisher by means of a push message. Once the publisher receives the subscription request, she can accept or decline the subscription. Whatever is the decision, it is finally notified to the application server.



(a) A user either a publisher or a subscriber wants to login into the mobile application

(b) A subscriber wants to receive alerts from a particular publisher

(c) A publisher produces and alert that is notified to all their subscribers

(d) A publisher wants to revoke the subscription of a particular subscriber

Figure 6.1: Different scenarios covered by the prototype system

- **Scenario C:** a publisher produces an alert, which must be notified to all their subscribers. The mobile application notifies the alert to the application server, which in turn sends a request to the GCM service with the information about the alert and the subscribers that must be notified. Finally, the GCM service sends to each subscriber a push message notifying about the alert. Optionally, the mobile application can request to the application server the previous alerts from the publisher at any time.

- **Scenario D:** a publisher is no longer interested in sending alerts to a particular subscriber. The publisher revokes the subscription in the mobile application. This action is notified to the application server, which marks the subscription as canceled.

The four above-mentioned scenarios involve different interactions between the mobile application, the application server and the GCM service. To make them work, the application server offers RESTful web services that can be accessed by a mobile application, which are listed in Table 6.1. Notice that the USER_ID, PUB_ID, and SUB_ID in the Table refer to the identifier of the particular user, publisher, or subscriber accessing the service.

Table 6.1: Services offered by the application server to the publisher and the subscriber

	Method	Endpoint	Description	Scenario
User	POST	/user/getUser	Notify the GCM id and receive the demographic data	A
	GET	/user/USER_ID /getContactsAccepted	Get the list of contacts with an accepted subscription	A
	GET	/user/USER_ID /getContactsPending	Get the list of contacts with an unconfirmed subscription	A
Publisher	POST	/publisher/PUB_ID /confirmSubscription	Accept a subscription request	B
	POST	/publisher/PUB_ID /revokeSubscription	Reject a subscription request or cancel an active subscription	B, D
	POST	/publisher/PUB_ID /notifyAlert	Notify an alert	C
Subscriber	POST	/subscriber/SUB_ID /subscribe	Request a subscription	B
	GET	/subscriber/SUB_ID /alertsByUser	Get the alerts of a particular publisher	C

6.1.1 Use Case for Monitoring Diabetes

We define a use case for the prototype system that illustrates the use of the four scenarios described in the previous section. In this use case we monitor diabetes mellitus, and in particular the blood glucose levels. To achieve this goal we define an agent with a Java mind that can produce the following alerts:

- Hypoglycemia, when the glucose is below 3.9 mmol/L.
- Hyperglycemia, when the glucose is above 7.2 mmol/L.

Two different users are involved in the use case: the patient Jane Doe affected by Diabetes Mellitus (DM) Type II, which has the role of publisher, and her medical doctor John Doe that acts as a subscriber. Initially the doctor is not subscribed to the patient's alerts.

Jane Doe has just taken a blood sample to check her glycemia before having breakfast. The glucometer reports her 3.8 mmol/L. After login into the mobile application with her username and password, she submits the value and the time of the measurement (Figure 6.2a). The agent analyzes the value and tells her that she has an hypoglycemia alert. The alert is also transmitted to the application server. The application server just stores the alert, as after checking Jane's subscribers it finds that nobody is subscribed to her alerts yet.

John Doe is now interested in receiving Jane's alerts. To ask her for the permission, he logs in to his mobile application and types her username (Figure 6.2b). The request is processed

by the application server, which contacts with the GCM service to send a push notification to Jane's smartphone. She receives John's petition, which she can accept or reject (Figure 6.2c). She decides to accept it, so that doctor John is now subscribed to her alerts. In the application's main screen he can see that she accepted his petition (Figure 6.2d). From here, he can navigate through a list of Jane's alerts.

Jane Doe takes another blood sample to check her gycemia level. The glucometer reports that her level is 7.4 mmol/L. She types the value in her mobile application as she did before. This time the agent reports her that she has an hyperglycemia. The alert is transmitted to the application server, which stores it in the database, and contacts with the GCM service to notify John about it. The GCM service sends a push notification to John's smartphone that is displayed in the notification area of his mobile device (Figure 6.2e).

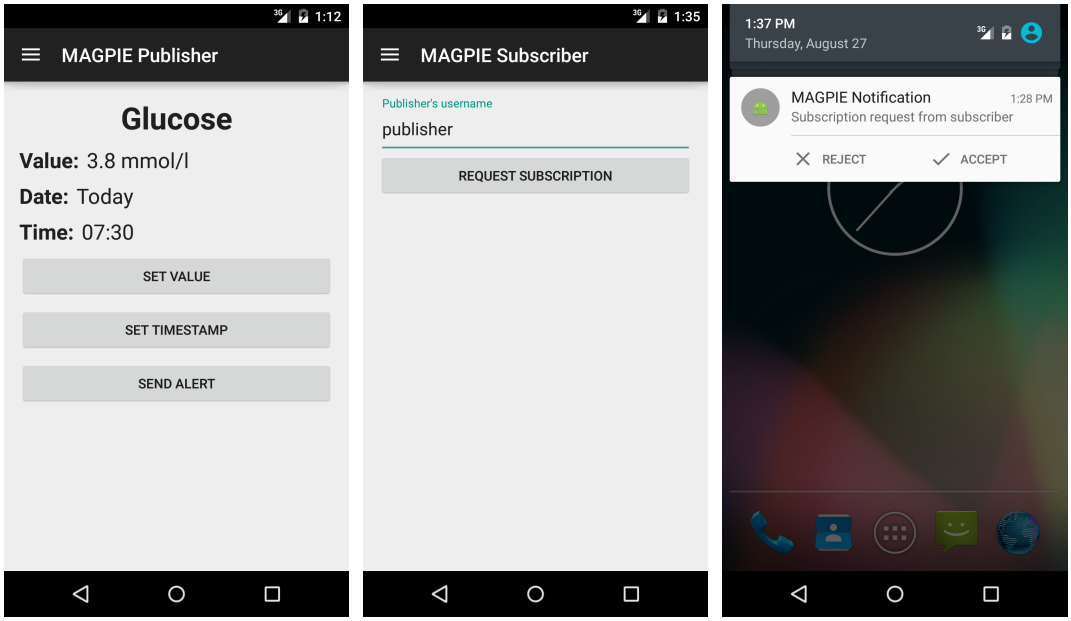
Jane Doe changed her medical doctor, so she is no longer interested in sending her alerts to John Doe. By long pressing on his name in the application's main screen a dialog appears where she can revoke his current subscription (Figure 6.2f).

6.1.2 Evaluation

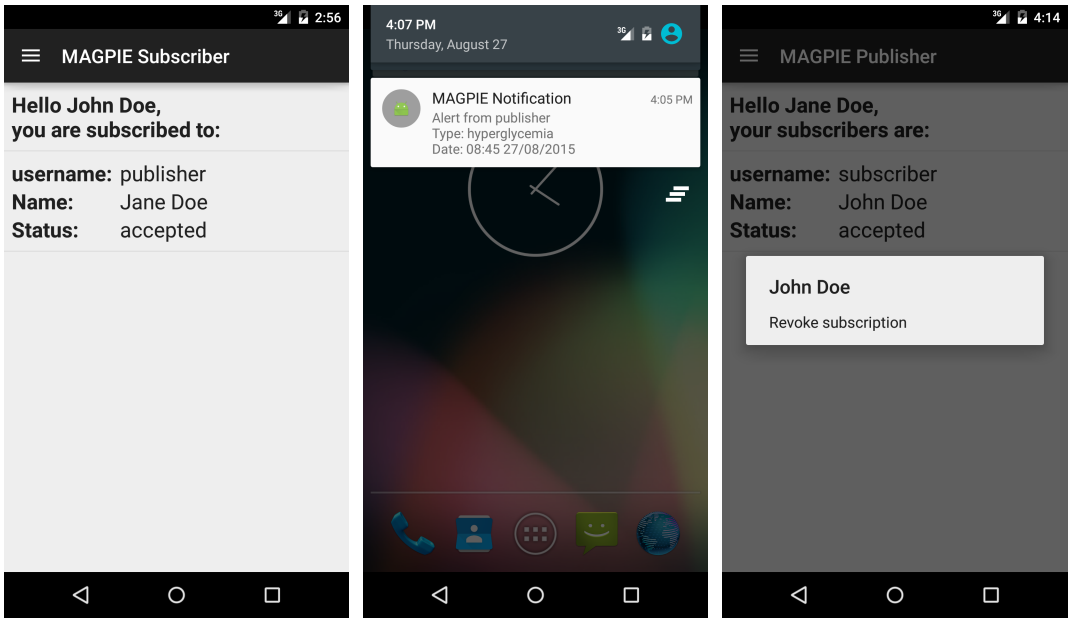
The prototype system described in this section is evaluated here in terms of the latency to deliver an alert, i.e. the time needed since the agent generates an alert until a subscriber is notified about it. Three different test scenarios are defined for the evaluation, which are shown in Figure 6.3: i) local scenario, where the publisher, the subscriber and the application server run on the same machine; ii) distributed scenario A, where the publisher and the subscriber run in one machine and the application server in a different one, with both machines connected in the same Local Area Network; iii) distributed scenario B, which differs from the previous one in that the machines are connected through Internet. Notice that in all the three test scenarios real Internet traffic is needed since the push mechanism relies in the GCM service, and that a single publisher and a single subscriber are considered. We used Android emulators to run the publisher and the subscriber, and the Apache Tomcat web server to run the application server. In the distributed scenario B the application server is deployed in the cloud using the Amazon Web Services [5].

Figure 6.4 shows the results of the experimental evaluation for the three different scenarios. The mean latency and the standard deviation in the Figure are computed over 100 repetitions per scenario. In the local scenario the mean latency obtained is 160.35 ms. This result increases 14.47 ms when passing to the distributed scenario A, and 65.26 ms with the distributed scenario B. The distribution of the latency over the repetitions is more concentrated for the local scenario, where the standard deviation of the latency is 56.6 ms, while in the distributed scenarios this value is more than the double.

The simulation scenarios presented above consist on one single publisher and one single subscriber. To have a clearer view on how the system behaves, a sweep in the number of subscribers should be performed. However, this has not been possible since we don't have a significant amount of Android devices nor powerful computers able to run multiple emulator instances.



(a) Report glucose measurements **(b)** Request a subscription **(c)** Notification requesting a subscription



(d) List of subscriptions **(e)** Notification of an alert **(f)** Revoke a subscription

Figure 6.2: Screenshots of the mobile application

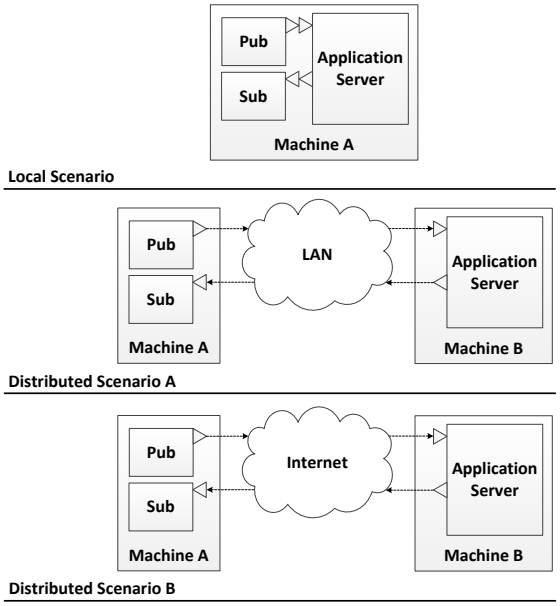


Figure 6.3: Test scenarios of the prototype system

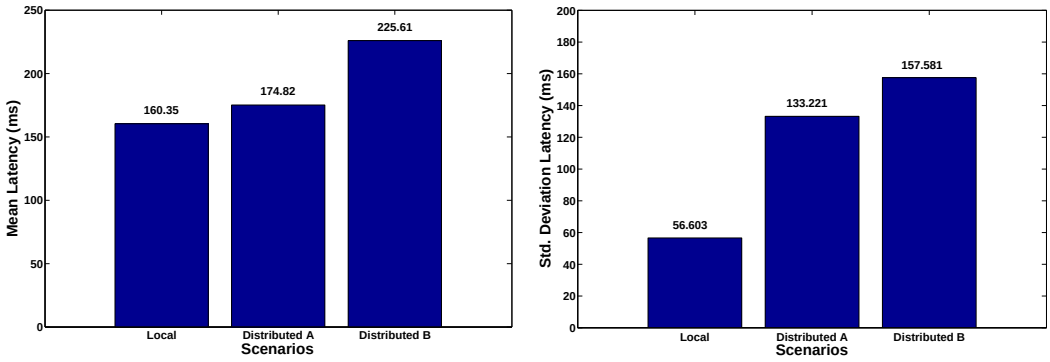


Figure 6.4: Mean and standard deviation of the delivery time in the different test scenarios

6.2 Sharing Data Between Institutions: The MOSAIC Protocol

Clinicians often need to compare the information collected from their patients with medical systems and clinical exams with information from similar patients in other places. This is needed for accurate diagnosis, prognosis and an effective management of diseases. Providing mechanisms to facilitate the access to remote worldwide distributed datasets becomes relevant to foster collaboration and knowledge sharing.

When all the ethical and legal regulations to protect the clinical data are satisfied a negotiation process for data exchange can start. A clinician may add some constraint and give access

to the data only if a certain set of conditions are satisfied. One typical constraint may be that another dataset is provided in return. Bilateral agreements between two clinical centers will not always solve all those constraints and involving a set of centers in the multilateral agreements for data exchange would increase the amount of data potentially accessible in the network.

This section describes the MOSAIC network protocol and analyzes its security. The goal of this protocol, which is based on MAS, is the finding of paths involving a set of nodes that all together can participate in a multilateral agreement for data exchange and knowledge sharing.

6.2.1 Scenario

The scenario where MOSAIC applies consists of a series of interconnected nodes each one with its local Data Mart and Agent Platform associated. The Data Mart of each node stores a specific data type cases classified by categories (e. g. cases of brain tumor types). Users can decide which type of cases from their local Data Mart they want to share with the rest of the nodes, or they can request the access to a certain type of cases shared by another node. The process for delivering to the users a certain data requested by them can be divided in four main stages:

- **Network exploration:** starts when the user selects a type of resource in which is interested. The resource can be found in several nodes, so for each option the different paths that allow access to the resource are explored.
- **Agreement Selection:** for every resource found, a path is selected among all the possible paths. The notification of the path selected is transmitted to the whole agents participating in the path.
- **Data Transfer:** after being received the notification of the path selected the data exchange between nodes starts.
- **Transaction Completion:** in the case that all nodes have received the data correctly the authorization for its use is transmitted, otherwise a disavowal message is transmitted.

A simplified example of the network exploration stage is the following:

- One of the users of a node A makes a petition to a node B the access to their shared cases of a specific category.
- The node B can allow directly the access to data or can require the delivery of some constraint to allow the access. Here the constraint is another category of the same type of data (e. g. a different case of brain tumor).
- If node A has the constraint required by node B a data exchange between them can be done. A possible bilateral agreement has been found.

- If node A can not deliver the constraint required by node B, then can search for it in a node C sharing this category of data. In the same way, node C can also allow the access directly or require the delivery of some constraint to exchange its data. In this way a recursive process starts and if it finds a solution, a possible multilateral agreement is found.

The exploration stage of the protocol ends with a possible agreement when the last node found shares its information freely, or when its constraint requested can be solved locally by the node which makes the request.

At the end of the process the node which started the network exploration has a set of different paths to access a specific resource of a different node, and has to choose one of them. The selection of a specific path implies the data transfer agreed during the network exploration. When a node receives the expected data then sends a confirmation to the requester node. When all the confirmations have been received the requester node sends the authorization for the use of data to the rest of nodes. At the end of the protocol all participating nodes in data transfer have obtained cases of data of their interest.

6.2.2 Protocol Agents

The negotiation process for data exchange is automated with the following agents:

- **MultiCast Contributor (MCC):** is activated by the user to share a specific category of data with the users from the rest of the nodes of the system, with or without requesting a constraint.
- **UniCast Contributor (UCC):** activated by the MCC to process a request received by a MultiCast Petitioner agent.
- **MultiCast Petitioner (MCP):** can be activated by the user to explore the network for searching a specific resource. It can also be activated by a UniCast Petitioner agent in order to try to solve the access constraint from a UCC when the constraint is not found in the own node.
- **UniCast Petitioner (UCP):** activated by a MCP in order to negotiate with an UCC the access to its dataset.
- **Yellow Pages (YP):** this agent is the service directory where all the active MCCs of the network are registered.

For the sake of clarity in the rest of the chapter the set formed by a MCC agent with one of their UCC agents will be referred as a Contributor, and the set formed by a MCP agent and one of their UCP agents will be referenced as a Petitioner.

A more detailed description of all these agents can be found in [96].

6.2.3 Possible Threats and Attacks

Assuming that the protocol does not have any security mechanism, a first analysis of the possible threats and attacks concludes that the external attackers can

1. perform passive attacks, for example monitoring the activity of the agents may provide confidential information about how many information the nodes exchange, how often, etc.

2. perform active attacks as

   (a) Denial of Service attack to one node or to the directory service. This type of attack against the directory service can disable communications between nodes.

   (b) identity spoofing of either a node or the directory service. In this case spoofing a node could be used to run malicious agents trying to exploit security holes of legitimate agents of the system.

   (c) the elimination of messages exchanged by the agents trough the network which may alter the interaction between them.

   (d) changing the content of the messages exchanged by the agents, for example changing the type of data requested by one agent.

   (e) the fabrication and insertion of messages into a communication with bad proposes, for example a message for deleting an agent.

   (f) unauthorized access to data stored in the nodes, which may compromise its integrity and confidentiality.

Regarding to the system users it must be taken into account that they can

1. misuse the system or misbehave, for example

   (a) rejecting deliberately all the access requests to their information shared.

   (b) sharing information in the network which they are not allowed to share. In this case the rights of the information can be from other user of the same node or from a user of another node.

   (c) sharing data that do not match with its description.

2. perform active attacks using malicious agents for its own profit.

3. try to increase the reputation of their agents or decrease the reputation of the agents from other nodes. In this case various options are possible as

   (a) try of modify the reputation value directly.

(b) to perform a Sybil attack with the creation of false nodes to increase their own reputation indirectly.

(c) the collusion of two different nodes trying to increase their own reputation.

(d) not updating properly the reputation from the user who has sent the data once the data exchange process has finished, thus producing a disadvantage for the latter.

4. alter the data exchange process

(a) not initializing data transmission when it is required.

(b) transmitting random information.

(c) not notifying the correct reception of data.

In all these cases the protocol has to grant that the data sent can be only accessed and used if the data received matches with the agreement done during the negotiation stage.

6.2.4 Security Architecture of the Protocol

Based on the preceding attacks the security architecture of MOSAIC (Figure 6.5) has the following blocks:

- **Transmission Protection:** guarantees the integrity and confidentiality of the data sent through the network, and the authenticity of the communicating entities.
- **Node Protection:** rejects any unauthorized external access to the internal network's node.
- **Access Control:** guarantees that the users use correctly the system.
- **Metadata Protection:** guarantees that agents and nodes reputation is not fraudulently altered.
- **Data Ownership Protection:** its goal is to ensure that no user act against the distribution of data.
- **Fair Exchange:** ensures that in every exchange of data both parts receive the expected data by offering a no repudiation service.
- **Agent Design:** defines the behaviors that the agents must have when receiving unexpected messages.

The transmission protection module guarantees the privacy and the integrity on the communication messages exchanged by the agents of the protocol. This goal is reached by ciphering and using hash functions on the messages. This is for both communications between agents of different nodes and communications between agents of the same node, as agents of different

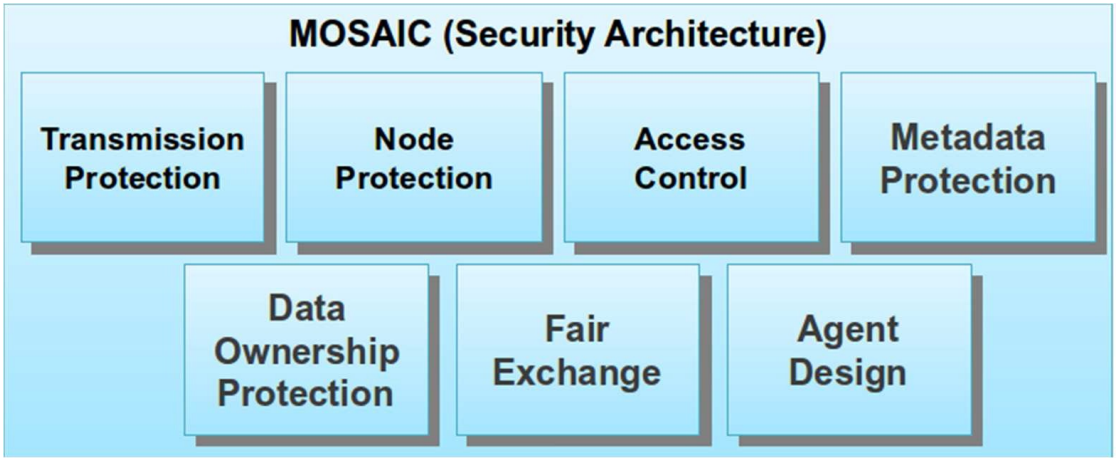


Figure 6.5: Components of the security architecture of *MOSAIC*

hosts from the same node may require communication. This block is also responsible to provide the identity of the node to the rest of the nodes of the network and to ensure the identity of the communicating nodes.

The node protection block is responsible to protect the node against any unauthorized access to its internal network. Specially it protects the database where data is stored, as an unauthorized access to it could compromise the confidentiality and integrity of the data stored. For this purpose each node has its own firewall and intrusion detection system.

The access control determines which users can use the system and which actions they can perform. For example, a user may be authorized to realize queries to exchange information with other nodes but not to share information with the nodes, or a user may not be authorized to accept a request to exchange data.

The metadata protection block has the goal to protect the reputation of the agents of every node. The reputation must not be fraudulently modified and it must be updated properly. This is an essential block in the security architecture of MOSAIC as the reputation of the agents is a parameter to consider when decide for a specific negotiation path.

The data ownership protection detects if the data obtained on an exchange was from the own node. The fact that the nodes exchange information does not imply that then are allowed to share it. To follow easily the date this block logs every information exchange (date, receiver node, uses who did the request, etc), and also introduces fingerprinting techniques to data.

The fair exchange block guarantees that when a negotiation agreement is accepted, if this is finally selected to make the data exchange, the data received is correct and only in this case is possible to access and use the data that was sent. To avoid and protect the nodes from dishonest behaviors this block provides a no repudiation service.

The last block, the agent design, is related with the behaviors of the agents according to the messages they receive. An agent must react properly when an unexpected message is received,

and its design must avoid flaws that can be used by malicious agents.

Finally it must be taken into account that data shared by nodes only contains the relevant information from a medical point of view. It is assumed that data is completely anonymized in such a way that its ownership lies to the doctor who generated it and decided to share. That means that data shared has lost its link with the person whom was obtained.

6.2.5 Analysis for Fair Exchange and Malicious Behaviors from the Nodes

Some of the blocks of the security architecture of MOSAIC have solutions that have been widely discussed in literature. In this section the fair exchange problem and a possible malicious behavior from the nodes are introduced.

Fabrication of Management Messages

During the network exploration stage the agents from different nodes exchange a series of messages. Such exchange allows to determine which category of data has to send the Petitioner of one node to the Contributor from another node in order to receive the shared data. When such data is not available in the Petitioner's node, the Petitioner creates a new Petitioner to find it. At the end of this process there can be lots of paths to get the data that a user has requested. One of these paths has to be selected to initialize the data transfer. The initial Petitioner (the one activated by the user) is the agent who selects the path and notifies the initialization of data exchange stage to the rest of the Petitioners from the selected path. Figure 6.6 shows an example of the data exchange process. In this example the network exploration stage ends when a Petitioner from node *i* has to send a category of data that the own node has. This stage can also end if a Contributor offers the data to node *i* without constraints. As we can see all the Petitioners created in a negotiation chain (except the one created by the user) act as intermediaries between the Contributors participating in the agreement.

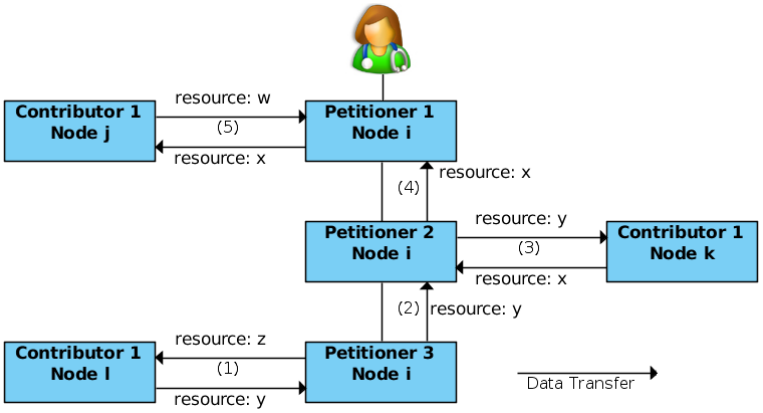


Figure 6.6: Data transfer process

The data exchange process can end with a complete and satisfactory transaction by all the agents, or with some error in one or several nodes. A transaction is considered as a satisfactory when all the Contributors and the initial Petitioner have received the data agreed during the network exploration stage. In this case every Contributor sends an ACK message to the initial Petitioner, or a NACK message otherwise. After receiving all ACKs the initial Petitioner sends to the rest of the agents a COMMIT message to confirm the authorization to use the data received. In the case that some ACK is not received or a NACK is received, the initial Petitioner sends a ROLLBACK message to the rest of the agents that unauthorizes the use of received data.

In this process there can be malicious behaviors that modify the normal operation of the protocol. One example can be a Contributor sending a NACK instead of an ACK, or a Petitioner sending ROLLBACK instead of COMMIT. To address these problems non repudiation mechanisms must be introduced into the protocol. These mechanisms can be used in the data transfer stage according to the trust degree between the parts involved in the exchange. In addition, the agents' reputation plays a fundamental role in the path selection. Using the agent reputation one can avoid the interaction with agents that have had this kind of behavior in previous negotiations. So every time a data exchange has finished as expected, the participating users should update the reputation of the agents from whom have received data. In this way when a Petitioner has to select a path, is more desirable to choose paths with agents having high reputation than paths with agents having low reputation or who have had unexpected behaviors in the past. The reputation can be also useful on the network exploration stage. When a Petitioner makes a request to a Contributor, the Contributor can directly reject the request taking into account the reputation of the final recipient. On the other hand, the reputation manifests the degree of collaboration of the agents so that those agents that do not encourage collaboration are harmed in long run.

Malicious Use of Loops in the Exploration Stage

During the exploration stage of the protocol there is the possibility that a loop appears between two nodes. This situation happens when a Petitioner from one node makes a request to a Contributor of another node which had already received a request from another Petitioner belonging to the same negotiation branch. As shown in Figure 6.7 a loop can appear when a Contributor has as constraint a resource that was already asked by a Petitioner.

The management done by a Contributor when this situation arises is not to impose any constraint to the request of the second Petitioner. With such behavior all nodes are benefited as now a multilateral agreement is possible.

The loop situation implies that both Petitioner and Contributor must implement extra functionality to handle this situation adequately. This management is done by processing a Request packet by both agents. The Request packet is a token that identifies a request, and it has the following parameters:

- The node ID from the requesting node

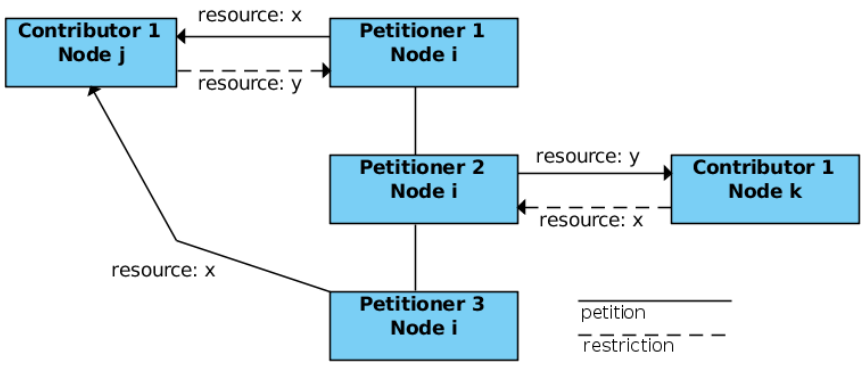


Figure 6.7: Loop example during the negotiation stage

- The ID of the first MCP from the negotiation chain (the one activated by the user)
- The ID of the negotiation branch

When a user creates a Petitioner agent in order to search a specific resource into the network, the Petitioner agent creates a new Request token. The first two fields of the token are filled with its corresponding values and the third with a random number. As the Petitioners explore all the possible agreements with the Contributors of the network a Petitioners tree is built, and each branch of this tree is an agreement path. Each new Petitioner agent receives the Request token from its creator agent and adds a new random number to the third field of the token. In this way when the network exploration stage ends, each Petitioner in a leaf of the tree has an array that uniquely identify its agreement path. Each Request token generated by a Petitioner agent is sent into the request made to the Contributor agent. The Contributor agent has a pool with all the active requests received until the moment. Each new Request token received is compared with the ones in the pool. If a new Request token is detected to be the same from one of the pool then the Contributor offers its data without any constraint.

Assuming an honest behavior of the agents the structure of the Request token provides security in the loops that can appear during the exploration stage of the protocol:

- The node ID field prevents that two petitions of two agents from different nodes be interpreted as the same.
- The ID of the first Multicast Petitioner field prevents that two petitions of two agents from the same node but from different trees be interpreted as the same.

The use of random numbers to identify each branch provides also extra security. As the ID from each Petitioner agent is unique it could be used to identify the branches, but using random numbers instead the Contributors do not receive information about the agents from the node they are negotiating.

This scheme for manage the loops has two different vulnerabilities that can be used by malicious agents. The module of agent design is to detect these malicious behaviors during the exploration stage of the protocol. Figure 6.8 and Figure 6.9 shows both examples.

In Figure 6.8, the Petitioner from one node makes a request to a Contributor from another node in order to get its shared resource. The Contributor notifies its constraint and then the Petitioner creates a new Petitioner to solve the constraint. The new Petitioner instead of trying to solve the constraint makes a second request to the same Contributor directly. In this way the Contributor would send its resource in data transfer stage without receiving the constraint requested. To solve this problem, in a loop case the Petitioner must notify who is the final node whom will receive the data. Taking this into account if the final receiver in a loop case is the same node as the petitioner node the request can be rejected directly. This means that a Petitioner that would like to cheat has to find another Contributor that shares a resource from the same type of the constraint requested from the Contributor being attacked. The resource from the Contributor found can be offered with or without delivering a constraint which do not modify the normal operation mode of the protocol.

In the proposed solution there is the possibility that a Petitioner and a Contributor from two different nodes agree to do this attack. In such case one must take into account that the constraint notification is done once the request is received so there is no possibility for a Petitioner to know the constraint in advance, and thus the following conditions apply:

- The attacker Contributor has to have the constraint requested by the attacked Contributor.
- The attacker Contributor must be created after the request realized by the attacked Petitioner.

These facts can be used by the attacked Contributor to detect if it is being attacked. The YP can provide to the attacked Contributor the timestamp when the attacker Contributor was set up into the network.

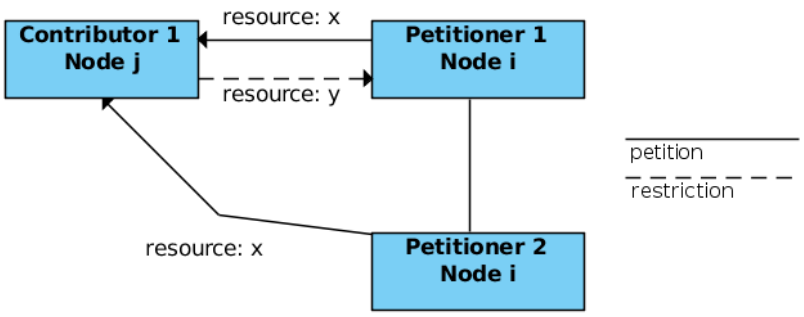


Figure 6.8: Example of a illicit request by a Petitioner agent

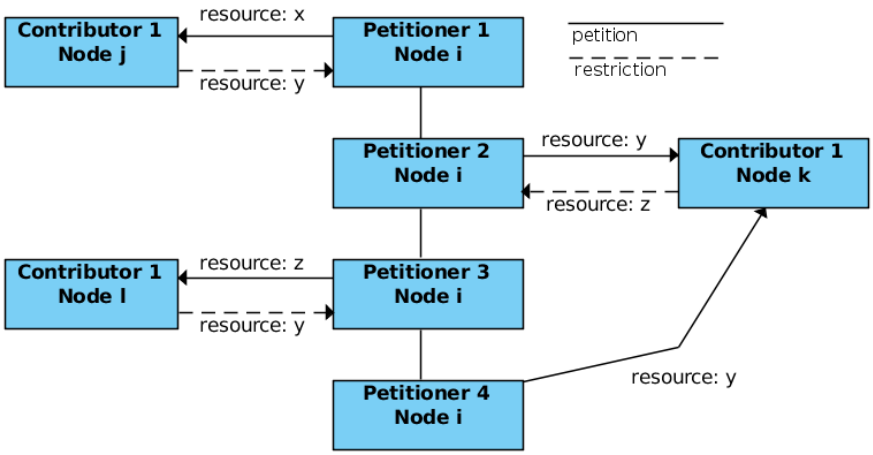


Figure 6.9: Example of licit request. In this case the Petitioner 4 can misuse by sending the resource 'y' to Petitioner 1

In Figure 6.9 a loop situation also arises, but this scenario is different from the previous one. In this case the second request received by the Contributor from node *k* is lawful as it comes from the constraint of Contributor from node *l*. In this case it is not possible to detect anything strange during the network exploration stage. However Petitioner 4 can misuse if it sends the resource "y" to Petitioner 1 instead of sending to the Contributor of node *l*. In such case as the Contributor of node *j* is not the final receiver, the Contributors on the middle of the negotiation chain are in disadvantage position (Contributors from nodes *k* and *l*). Taking this into account the data sent over the network must be ciphered with the public key of the final receiver in case of using public key cryptography, or ciphered with a symmetric key shared by the initial sender and the final receiver in case of using symmetric key cryptography.

Chapter 7

Conclusions and Future Work

"There is no real ending. It's just the place where you stop the story."
Frank Herbert.

Contents

7.1 Conclusions

Pervasive healthcare consists on applying pervasive computing technologies to assist in healthcare related issues. Traditionally, healthcare services are delivered in a centralized way where specialized medical doctors treat acute patients in hospitals. The pervasive healthcare research field promotes a more distributed delivery model of healthcare services, where the services are focused on the patients. This distributed model implies continuous monitoring of patients, without being restricted by physical locations, and involving them in the management of their illness and promoting their own well being too. Thus, the healthcare delivery model becomes more

proactive and preventive, rather than being reactive. This shift in the model is expected to have an economic impact in the healthcare expenditures, due to the fact that life expectancy is increasing and with it also increases the population affected by chronic illness. Thereby, these people affected by chronic conditions need long term solutions that improves their quality of life.

At the beginning of the thesis we identified four main application environments of pervasive healthcare technology, which are Personal Health Systems (PHSs), Ambient Assisted Living (AAL), pervasive computing for hospitals, and persuasive technologies. We have seen that the development of these systems have implicitly different technological challenges that must be faced like its interoperability, scalability, and the formalization of the medical knowledge. In this thesis we have focused on providing solutions that tackle these challenges, applying them to a PHS to manage Diabetes Mellitus (DM).

The first contribution of this dissertation consisted on a literature review to have a clear picture of the state of the art on these topics. We have seen the importance of developing interoperable systems, and different types of standards to support interoperability in the healthcare domain, which can be classified in standards for messaging, terminology standards and standards for Electronic Health Records (EHRs). Regarding scalability, we have seen different strategies to achieve scalable systems like distribution of tasks, early processing of information, and provide more powerful hardware. Though most of the reviewed systems just claim being scalable but do not measure how scalable they are. Respect the other challenge faced in this thesis, the formalization of the medical domain, we have seen that there are different categories of methods that can be applied. These categories include supervised learning, unsupervised learning, probabilistic logic, fuzzy logic, ontological reasoning and temporal reasoning. We can conclude from this review that most of the systems found in the literature do not consider interoperability nor scalability in their development. Moreover, decision support techniques in medicine do not involve directly the expertise of medical doctors.

The next contribution in this thesis has been to deal with the interoperability of our PHS. The approach consisted on using the Clinical Document Architecture (CDA) standard to build EHRs that capture the state of the patient in a certain time window. We applied this solution to the use case of Gestational Diabetes Mellitus (GDM), and in line with our strategy to provide scalability, these documents were generated in the Tier 2 of the system, and sent over the network to the Tier 3 of the PHS. We have seen that these documents can be generated automatically by combining different prebuilt XML templates, whose variable values can be filled by using XPath expressions. For the codification of the different clinical terms (physiological parameters, symptoms, medications, and alerts) we used vocabulary standards like SNOMED CT, LOINC, ICD, and ATC. However, we found that there is a need to extend the CDA specifications in order to consider the patients' involvement in self management diseases like DM. In our case we have two types of insulin doses, the one taken by the patient and the one prescribed by the doctor, while the CDA standard only specifies one attribute to encode specific dosages. We solved this issue by adding the `<dosePrescribed>` element to model the insulin dose prescribed by the

doctor, and we used the `<doseQuantity>` element of the standard to specify the insulin dose taken by the patient. In relation with the interoperability, we analyzed two different strategies for generating the documents with the aim of minimizing the data sent through the network and extend the battery life of the mobile device. In particular, the strategies were based on generating a document when there is a new alert, or do it at the end of each day. The results showed that the total data sent through the network was 87% more when using the daily-based strategy in comparison with the alert based-strategy. This is due to the fact that every document must have a header part, which has a big weight in the total size of the document.

The third contribution of the thesis consisted on achieving a scalable PHS. We followed a strategy that consisted on performing the computations needed for the monitoring task on the Tier 2 of PHS, by taking advantage on the performance of current mobile devices available in the market. In this context, we developed the MAGPIE agent platform. This agent platform is conceived as a framework to provide intelligence to PHS, where we model concepts from Multi-Agent Systems (MAS) that are integrated with the Android operating system. The MAGPIE agent platform is a key component of our PHS and is linked somehow to the strategies for providing interoperability and the formalization of the medical knowledge. From one side, the CDA documents discussed before can contain alerts produced by the agents running in the platform. From the other side, the agents are able to run monitoring rules defined by the medical doctors. To evaluate our distributed strategy where the alerts are produced in Tier 2, we performed a simulation to compare it with a centralized strategy where alerts are produced in Tier 3 of the system, a shared component for all the patients. The results showed that when the number of users increases, the latency has a flat response with the MAGPIE approach, while with the centralized approach the latency increases linearly. Thus, we recommend to place the reasoning part of PHS in Tier 2, close to the patient, to achieve scalable systems.

In line with the MAGPIE agent platform, the fourth contribution of the thesis consisted on the formalization of the domain knowledge. We followed a temporal reasoning approach based on Event Calculus (EC). In this approach we modeled temporal patterns of physiological values, which are in fact monitoring rules understandable by agents with a Prolog mind. We distinguished between two types of rules: complex and sequential, where the difference between both is that in complex rules does not matter the order in that the events happen, while in the sequential rules it does. The particularity of our proposal is that these rules can be specified in a graphical way, which makes it possible for the medical doctors to define them autonomously. Another point that must be outlined here is that the way we face the formalization challenge allows us to have a reconfigurable PHS at runtime. We evaluated the proposed method by defining some temporal patterns and trying to find them in a real dataset of patients affected by DM Type II. Though the selected patterns reported unwanted situations, it is important to stress that our proposal can be used to report good conditions, like having blood pressure values in a normal range the seven days of a week.

The last contributions of this thesis are related with sharing the data collected with a PHS

to improve the outcomes obtained with the use of this technology. We presented two different ways of doing that. First, we integrated the use of MAGPIE in a Distributed Event-Based System (DEBS), where other patients, relatives and doctors can subscribe to the alerts produced by the patient. We built a prototype system that proves the feasibility of this approach when using Google Cloud Messaging (GCM) to send push messages for the notification of the subscriptions requests and the alerts. Second, we presented the MOSAIC protocol to exchange data between different nodes, which can be healthcare institutions. By analyzing the possible threats to this protocol, we derived a set of considerations to make it secure.

7.2 Future Work

7.2.1 Self Protected CDA Documents

In the PHS discussed in this thesis, the security to transmit the data over the network is on the communication channel. The CDA documents do not have any kind of protection in the endpoints. To do a step further in terms of security, the security services can be embedded inside the CDA documents, so that they become a self protected entity of data.

7.2.2 Intelligent Strategy for Generating CDA Documents

We presented two different strategies to decide when the generated CDA documents must be sent. The next step would be to learn the user profiles and make the decision based on these profiles to find the right balance between the times that the documents are sent and the total data sent through the network.

7.2.3 Integration of Interoperability in MAGPIE

In the thesis we provided interoperability at the application level. An enhancement to the MAGPIE agent platform could be to integrate interoperability on it, and use newer standards like FHIR.

7.2.4 Interface for Rule Learning

In the web application presented in the thesis, the rules are defined manually by the medical doctors. Another interesting approach could be that the system learns automatically temporal rules from the data collected, and present them to the doctors.

7.2.5 Topic Subscription in DEBS

In the prototype of the DEBS with MAGPIE, the subscribers are notified about all the alerts produced by the patient. The next step here would be that the subscriber can filter the alerts from different categories.

7.2.6 Intelligent Path Selection in MOSAIC

The MOSAIC protocol presented in this thesis has the drawback that the network must be flooded to find all the possible paths that end up in an agreement. This issue makes this protocol inefficient and unscalable. This can be mitigated by introducing intelligent algorithms to decide which is the best branch to explore, or the more likely to have an agreement.

Appendices

Appendix A

CDA Templates

Contents

This appendix is related to Chapter 3 and details the different XML templates and XPath expressions used to build the CDA documents described in that chapter. The question marks in the templates correspond to the values that are selected and filled with the specified XPath expressions.

A.1 Header

1. //id/@extension
2. //effectiveTime/@value
3. //patientRole/id/@extension
4. //author/time/@value

5. //assignedAuthor/id/@extension

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cda.xsl"?>
<ClinicalDocument xmlns="urn:hl7-org:v3"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="urn:hl7-org:v3 CDA.xsd">

    <!--**************************************************
      CDA Header
    **************************************************-->
    <typeId root="2.16.840.1.113883.1.3" extension="POCD_HD000040"/>

    <!-- UID of the document -->
    <id root="?" extension="?"/>

    <!-- Type of document -->
    <code code="51855-5" codeSystem="2.16.840.1.113883.6.1"
    codeSystemName="LOINC" displayName="Patient Note"/>

    <!-- Creation time of the document -->
    <effectiveTime value="?"/>

    <!-- Confidentiality code, N=Normal -->
    <confidentialityCode code="N" codeSystem="2.16.840.1.113883.5.25"/>

    <!-- Person whose chart this document belongs to -->
    <recordTarget>
        <patientRole>
            <!-- ID of the Patient -->
            <id root="?" extension="?"/>
        </patientRole>
    </recordTarget>

    <!-- Humans and/or machines that authored the document -->
    <author>
        <!-- Date of the document -->
        <time value="?"/>
        <!-- Author of the document -->
        <assignedAuthor>
            <id root="?" extension="?"/>
        </assignedAuthor>
    </author>

    <!-- Organization that is in charge of maintaining the document -->
```

```
    <custodian>
        <assignedCustodian>
            <representedCustodianOrganization>
                <!-- ID of the custodian organization -->
                <id root="?"/>
            </representedCustodianOrganization>
        </assignedCustodian>
    </custodian>

    <!--**********************************************************
      CDA Body
    **********************************************************-->
    <component>
        <structuredBody>
        </structuredBody>
    </component>
</ClinicalDocument>
```

A.2 Body Section

1. //component/section/code/@code

2. //component/section/code/@displayName

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <component>
        <section>
            <code code="?" codeSystem="2.16.840.1.113883.6.1"
             codeSystemName="LOINC" displayName="?"/>
        </section>
    </component>
</container>
```

A.3 Blood Pressure Entry

1. //entry/observation/effectiveTime/@value

2. //entry/observation/entryRelationship/observation[code/@displayName="Systolic BP"]
/value/@value

3. //entry/observation/entryRelationship/observation[code/@displayName="Diastolic BP"] /value/@value

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <entry>
        <!-- Blood Pressure -->
        <observation classCode="OBS" moodCode="EVN">
            <code code="251076008" codeSystem="2.16.840.1.113883.6.96"
            codeSystemName="SNOMED CT" displayName="Cuff blood pressure"/>
            <!--Timestamp of the measurement -->
            <effectiveTime value="?"/>
            <entryRelationship typeCode="COMP">
                <!-- Systolic blood pressure -->
                <observation classCode="OBS" moodCode="EVN">
                    <code code="271649006"
                    codeSystem="2.16.840.1.113883.6.96"
                    codeSystemName="SNOMED CT" displayName="Systolic BP"/>
                    <value xsi:type="PQ" value="?" unit="mm[Hg]"/>
                </observation>
            </entryRelationship>
            <entryRelationship typeCode="COMP">
                <!-- Diastolic blood pressure -->
                <observation classCode="OBS" moodCode="EVN">
                    <code code="271650006"
                    codeSystem="2.16.840.1.113883.6.96"
                    codeSystemName="SNOMED CT" displayName="Diastolic BP"/>
                    <value xsi:type="PQ" value="?" unit="mm[Hg]"/>
                </observation>
            </entryRelationship>
        </observation>
    </entry>
</container>
```

A.4 Heart Rate Entry

1. //entry/observation/effectiveTime/@value

2. //entry/observation/value/numerator/@value

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <entry>
```

```
4         <!-- Heart Rate -->
5         <observation classCode="OBS" moodCode="EVN">
6             <code code="364075005" codeSystem="2.16.840.1.113883.6.96"
7              codeSystemName="SNOMED CT" displayName="Heart rate"/>
8             <!--Timestamp of the measurement -->
9             <effectiveTime value="?"/>
10            <value xsi:type="RTO_PQ_PQ">
11                <numerator value="?"/>
12                <denominator value="1" unit="min"/>
13            </value>
14        </observation>
15    </entry>
16 </container>
```

A.5 Blood Sugar Entry

1. //entry/observation/text
2. //entry/observation/effectiveTime/@value
3. //entry/observation/value/@value

```
1  <container xmlns="urn:hl7-org:v3"
2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3     <entry>
4         <!-- Blood Sugar -->
5         <observation classCode="OBS" moodCode="EVN">
6             <code code="302789003" codeSystem="2.16.840.1.113883.6.96"
7             codeSystemName="SNOMED CT"
8             displayName="Capillary blood glucose measurement (procedure)"/>
9             <!-- Period of the day respect the meals -->
10            <text></text>
11            <!--Timestamp of the measurement -->
12            <effectiveTime value="?"/>
13            <value xsi:type="PQ" value="?" unit="mmol/L"/>
14        </observation>
15    </entry>
16 </container>
```

A.6 Weight Entry

1. //entry/observation/effectiveTime/@value

2. //entry/observation/value/@value

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <entry>
        <!-- Weight -->
        <observation classCode="OBS" moodCode="EVN">
            <code code="363808001" codeSystem="2.16.840.1.113883.6.96"
            codeSystemName="SNOMED CT" displayName="Body weight measure"/>
            <!--Timestamp of the measurement -->
            <effectiveTime value="?"/>
            <value xsi:type="PQ" value="?" unit="kg"/>
        </observation>
    </entry>
</container>
```

A.7 Symptom Entry

1. //entry/observation/code/@code
2. //entry/observation/code/@displayName
3. //entry/observation/effectiveTime/@value

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <entry>
        <observation classCode="OBS" moodCode="EVN">
            <code code="?" codeSystem="2.16.840.1.113883.6.3"
            codeSystemName="ICD10" displayName="?"/>
            <!--Timestamp of the symptom -->
            <effectiveTime value="?"/>
        </observation>
    </entry>
</container>
```

A.8 Medication Entry

1. //entry/substanceAdministration/text
2. //entry/substanceAdministration/effectiveTime/@value

3. //entry/substanceAdministration/effectiveTime/event/@code
4. //entry/substanceAdministration/doseQuantity/@value
5. //entry/substanceAdministration/dosePrescribed/@value
6. //entry/substanceAdministration/consumable/manufacturedProduct /manufacturedLabeledDrug/code/@code
7. //entry/substanceAdministration/consumable/manufacturedProduct /manufacturedLabeledDrug/code/@displayName
8. //entry/substanceAdministration/consumable/manufacturedProduct /manufacturedLabeledDrug/name

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <entry>
        <substanceAdministration classCode="SBADM" moodCode="EVN">
            <text></text>
            <!-- Timestamp of the injection in relation with the meals -->
            <effectiveTime xsi:type="TS" value="?"/>
            <effectiveTime xsi:type="EIVL" operator="A">
                <event code="?"/>
            </effectiveTime>
            <doseQuantity value="?" unit="IU"/>
            <dosePrescribed value="?" unit="IU"/>
            <consumable>
                <manufacturedProduct>
                    <!-- Insulin code and name -->
                    <manufacturedLabeledDrug>
                        <code code="?" codeSystem="2.16.840.1.113883.6.73"
                        codeSystemName="ATC" displayName="?"/>
                        <name></name>
                    </manufacturedLabeledDrug>
                </manufacturedProduct>
            </consumable>
        </substanceAdministration>
    </entry>
</container>
```

A.9 Alert Entry

1. //entry/observation/code/@code

2. //entry/observation/code/@displayName
3. //entry/observation/effectiveTime/@value
4. //entry/observation/value/@code
5. //entry/observation/value/@displayName

```
<container xmlns="urn:hl7-org:v3"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <entry>
        <observation classCode="OBS" moodCode="EVN">
            <code code="?" codeSystem="2.16.840.1.113883.6.96"
            codeSystemName="SNOMED CT" displayName="?"/>
            <effectiveTime value="?"/>
            <value xsi:type="CD" code="?"
            codeSystem="2.16.840.1.113883.6.96"
            codeSystemName="SNOMED CT" displayName="?"/>
        </observation>
    </entry>
</container>
```

Appendix B

MAGPIE: API Guide

Contents

The purpose of this appendix is to be a standalone API guide that complements Chapter 4 from a practical point of view. This appendix explains how to use the MAGPIE agent platform when developing an Android application. The source code of the platform can be found in

`https://github.com/aislab-hevs/magpie`

This API guide assumes that the reader is familiar with Android developer. If it is not the case, refer to the online documentation first

`https://developer.android.com/sdk/index.html`

B.1 Application Fundamentals

MAGPIE provides a framework to run agents on Android devices. It has been though as an agent platform to be used for the development of Personal Health System (PHS). This means that the framework expects as input values, events representing physiological values from the patient. These events are internally processed by the agents, which produce alerts according to

the behaviors of the agent. How the alerts must be processed when they have been created , i.e. display a notification to the patient, send the alert to a web server, etc. is a specific task that may vary from different PHSs, and as such depends on the application developer. Internally, the framework is divided in two layers.

- **Conceptual Level:** this level models concepts from Multi-Agent Systems (MAS) such as agents and an agent environment. The `Environment` class mediates the interactions between the agents and the events, which can come directly from the patient through the User Interface (UI) or from a sensor. The textttEnvironment class is a transparent component for the application developer, and Chapter 4 offers a description about its implementation. Agents are the computing entities responsible to provide feedback on the status of the patient. They capture the medical knowledge for realizing this task, and as so they must be programmed by the application developer.
- **Android Integration Level:** this layer provides the functionality that integrates the Conceptual Level with the Android Operating System. There are two main components in this layer, the `MagpieActivity` class and the `MagpieService` class. The `MagpieActivity` is an Android `Activity`, and is the entry point for the events expected by the platform. This class must be extended by at least one of the application's activities in order to provide this functionality. The `MagpieService` is an Android `Service` that runs the `Environment`, and provides to the `MagpieActivity` all the operations that can be done on that. The `MagpieService` is transparent to the application developer, and is already declared in the framework's manifest. In order to interact to each other, the `MagpieActivity` bounds automatically to the `MagpieService`.

B.2 Framework Components

There are three main components that the developer must consider when using MAGPIE: the `MagpieActivity`, the events, and the agents. In big terms the agents and the events are instantiated from the `MagpieActivity`, which provides the methods for these components to reach the `Environment`. This section explains in more details every component, and how they interact to each other.

B.2.1 MagpieActivity

The `MagpieActivity` class is a key component in the MAGPIE agent platform. This class is an Android `Activity` with added functionalities, and provides the methods to interact with the `MagpieService` and the `Environment`. Under the hood it binds automatically to the `MagpieService` to provide all these functionality, and it must be extended by one of the application's `Activity`.

```
public class MyActivity extends MagpieActivity {
  ...
}
```

An important method from the `MagpieActivity` is `onEnvironmentConnected()`. This method is called automatically by Android when the binding to the `MagpieService` has finished. The purpose of this method is to instantiate and add the agents to the `Environment`. The former is explained in details in Appendix B.2.3, the latter is done by invoking the method `getService().registerAgent()`.

```
public void onEnvironmentConnected() {
  // Instantiate the agents
  ...
  // Add them to the environment
  getService().registerAgent(...);
}
```

The `MagpieActivity` has also the method `sendEvent()` to send an event from the `Activity` to the `Environment`.

B.2.2 Events

In MAGPIE an event object represents a physiological measurement. The base class for manipulating events is the `MagpieEvent` class, which has two main fields: a `long` timestamp, and a `String` type. The timestamp is automatically set to the system's clock when an object of this class is instantiated, but it can be changed to any other value with the method `setTimeStamp()`. This allows the reporting of events that are not necessarily happening at the moment that are reported. The field type is used by the framework to route the events to the proper agents interested on that particular type of event.

The `MagpieEvent` class can be extended by another class to provide custom implementation for events. However, the MAGPIE agent platform provides the `LogicTupleEvent` class, which already extends the `MagpieEvent` class in a practical way. This class represents a measurement in the following `String` form: name(arg1, arg2, ..., argN). Where the *name* is intended to be used for identifying the represented physiological parameter, e.g. glucose, blood pressure, weight; and the *arg* for the values. Another thing to consider about the `LogicTupleEvent` is that it has the *type* field set to the constant `Services.LOGIC_TUPLE`.

The `LogicTupleEvent` class provides methods to manipulate the `String` representation of the measurement.

- `getName()` $\Rightarrow$ returns the name set to the physiological parameter.
- `getArgunments()` $\Rightarrow$ returns a `List<String>` containing the values of the measurement.

To submit an event from a MagpieActivity to the Environment, the sendEvent() method can be used. As example, consider that a glucose event is sent from an EditText when a Button is pressed. From the Button event we can call the following method, where we pass the text of the EditText as a parameter.

```
private void sendGlucoseEvent(String value) {
  LogicTupleEvent lte = new LogicTupleEvent("glucose", value);
  sentEvent(lte);
}
```

B.2.3 Agents

Agents are the computing entities responsible of analyzing the events reported by the patient. They must model the medical knowledge to provide the right feedback to the patient and the medical doctor. This feedback can be in different forms i.e. a notification in the notification area of the device, a connection to a web service, etc. and is up to the developer to decide which is the best option that matches the purpose of the PHS being deployed. In order to do these tasks an agent must be placed into the agent Environment.

An agent is composed by two parts: a body and a mind. The body is the part of the agent that receives/produces events from/to the Environment. The mind is the component in charge of managing the agent's reasoning abilities. Agents with a Java mind can run behaviors. A behavior is a task carried out by an agent in response to an Event received from the Environment.

The base class for defining behaviors is the Behavior class, which must be extended in order to create a new behavior. A Behavior object has a name that is automatically set in the constructor, and three optional fields: i) the agent to invoke methods from the body/mind, ii) the Android Context class to interact with the UI, and iii) a priority number to order behaviors. These three fields can be set in the class constructor using its corresponding setter methods. The following code snippet shows an example on how to do this.

```
public class MyBehavior extends Behavior {

  // Fields of MyBehavior class

  public MyBehavior(Context context, MagpieAgent agent, int priority) {
    setContext(context);
    setAgent(agent);
    setPriority(priority);
  }

  // Methods of MyBehavior class
}
```

A Behavior object must implement two methods from the IBehavior interface.

- boolean isTrigered(MagpieEvent ev) ⇒ this method evaluates when the behavior is triggered.
- void action(MagpieEvent ev) ⇒ this method defines what the agent does with the event.

Two extreme cases arise from the method `isTriggered()`, which are when the method returns always either `true` or `false`. Returning always `false` means that the particular `Behavior` will be never performed by the agent. On the other side, returning always `true` means that the `Behavior` will be always triggered, no matter what event is received. Something in between would be to specialize a `Behavior` for particular events, e.g. glucose. In the case of using `LogicTupleEvent` objects as events, this specialization is straightforward by using the methods from this class, which were explained in the previous section. The following code snippet shows how.

```
@Override
public boolean isTriggered(MagpieEvent event) {
  LogicTupleEvent condition = (LogicTupleEvent) event;
  return condition.getName().equals("glucose");
}
```

Something important to consider when programming a `Behavior` is that the agents run in a different thread from the UI thread. Therefore, it is not possible to perform operations directly into UI components. In this case is when the use of the `runOnUiThread()` method from the Android `Context` class comes handy. As example, suppose that a `Behavior` MyBehavior is instantiated from a `MagpieActivity` MyActivity. In this case the implementation of the `action()` method can be something similar to the following code snippet.

```
@Override
public void action(MagpieEvent ev) {
  // Preprocess the event
  ((MyActivity) getContext()).runOnUiThread(new Runnable() {
    // Run method
  });
}
```

Given a set of behaviors already programmed, the steps to initialize an agent and monitor a patient are the following: i) instantiate the body, ii) instantiate the mind, iii) add the behavior(s) to the mind, iv) set the mind into the body, v) add the agent to the `Environment`.

(i) The class `MagpieAgent` must be used to instantiate an agent body. The constructor takes a variable number of `String` parameters. The first one is the name given to the agent, and the subsequent ones are the type of events that the agent is interested on. In the case of only using the `LogicTupleEvent` class, this can be specified with the constant `Services.LOGIC_TUPLE` provided by the framework, as in the following example.

```
MagpieAgent agent = new MagpieAgent("monitoring_agent", Services.
    LOGIC_TUPLE);
```

(ii) The class for a mind working with behaviors is the `BehaviorAgentMind`. This class must be extended, and must implement the method `executeBehaviors()` to specify the order in which the different behaviors must be executed. The framework provides two specific implementations of minds that are able to run behaviors: the `SequentialBehaviorAgentMind` that runs behaviors in the same order that they are added to the mind, and the `PriorityBehaviorAgentMind` that runs the behaviors according to its priority number. In the latter case, higher priority number means higher priority in the execution of the behavior.

(iii) To add a `Behavior` into the mind, the method `addBehavior()` must be used.

```
SequentialBehaviorAgentMind mind = new SequentialBehaviorAgentMind();
mind.addBehavior(new MyBehavior(this, body, 0));
```

(iv) Once the mind has all the desired behaviors, it can be registered into the body as follows.

```
agent.setMind(mind);
```

(v) To add the agent into the `Environment`, from a `MagpieActivity` the following method must be invoked, passing the agent as a reference.

```
getService().registerAgent(agent);
```

B.3 Sensors

The MAGPIE agent platform can work with Bluetooth sensors that measure physiological values like the heart rate. More specifically, this guide shows the integration with the Zephyr's BioHarness sensor. The goal of this integration is that events created in the Android application are related to the sensor's measurements. To achieve this, at the Android Integration Level the classes `SensorService` and `SensorHandler` take care about the different steps involved in that process, like connecting with the sensor and reading the measurements, and route them to the `Environment`. The details on the interaction taken place between the different components involved in that process are out of the scope of this API guide, as they are explained in Chapter 4 of this thesis.

From the application development perspective, two main classes are involved in the use of a sensor that are the `MagpieActivity` and the `SensorHandler`. The `SensorHandler` class must be extended to provide the specific logic to connect with the sensor. In the case of using a BioHarness sensor, the class `ConnectListenerImpl` is also needed.

To start and stop a connection with a sensor, the `MagpieActivity` class provides respectively the methods `connectToSensor()` and `disconnectSensor()`. These two methods can be called from anywhere in the `MagpieActivity`, and typically this action can be related to a user input event like pressing a button from the UI. In addition, the method `connectToSensor()` takes as argument the specific class extending the `SensorHandler` class. The following code snippet shows the example of a `Button`, whose `onClick` attribute value is `connectToBioHarness`, and the `BioHarnessHandler` class is the class extending `SensorHandler`.

```
public void connectToBioHarness(View view) {
  connectToSensor(BioHarnessHandler.class);
}
```

The class extending `SensorHandler` must implement the methods `onStartConnection()`, `onStopConnection()`, and `processSensorMessage()`. These three methods represent the three different actions that can be done to a sensor that are respectively connect, disconnect, and read values. The first two methods are called by the framework in response to the beforementioned methods in the `MagpieActivity` class. The third method is called every time that there is a new sensor reading.

The method `onStartConnection()` handles the logic to establish a Bluetooth connection with the sensor, and returns an `int` code, which represents the result of the connection, e.g. the Android device does not support Bluetooth, the Bluetooth is not active, the sensor is not paired with the Android device, etc. The codes used and the cases handled by this method are up to the application developer. The code number is finally forwarded to the `sensorConnectionResult()` method in the `MagpieActivity` class. The purpose of this method is to inform the user about the result of the connection. A last consideration to take into account in this connection process is that when a connection with a sensor has been established successfully, the method `connectToAgentEnvironment()` must be called, so that MAGPIE establishes a communication channel with the `Environment`.

In the same way, the method `onStopConnection()` handles the logic to disconnect the Android device from the sensor, which tipically involves the release of resources. This method is called behind the scenes by the MAGPIE framework when the method `disconnectSensor()` is called in the `MagpieActivity`. Table B.1 shows a summary of the methods described to establish/disable a connection with a sensor.

Finally, the method `processSensorMessage()` is called when the sensor produces a new reading. The goal of this method is to prepare the `MagpieEvent` related with the measurement that is eventually sent to the `Environment`. To achieve this goal with the BioHarness sensor, the class `ConnctListenerImpl` must be implemented. This class is from the BioHarness API, and

Table B.1: Methods involved in the connection/disconnection to/from a sensor

	Method	Return	Class	Purpose
Connection	connectToSensor()	void	`MagpieActivity`	Initialize the connection
	onStartConnection()	int	`SensorHandler`	Logic for the connection
	connectToAgentEnvironment()	void	`SensorHandler`	Tell to MAGPIE that a connection has been established
	sensorConnectionResult()	void	`MagpieActivity`	Inform the user about the result
Disconnect.	disconnectSensor()	void	`MagpieActivity`	Initialize the disconnection process
	onStopConnection()	void	`SensorHandler`	Terminate the connection and release resources

it is responsible for creating objects according to the streams of data coming from the sensor. To communicate with our `SensorHandler` class, named `BioHarnessHandler`, a reference must be passed to the `ConnectListenerImpl` class. This operation can be done in the constructor as shown in the next code snippet.

```
public class BioHarnessConnectListenerImpl extends ConnectListenerImpl {
  private BioHarnessHandler handler;

  public BioHarnessConnectListenerImpl(BioHarnessHandler handler) {
        super(handler, null);
        this.handler = handler;
  }
}
```

To create the objects from the data stream provided by the sensor, the `ConnectListenerImpl` class must implement the `Connected()` method. This method has the purpose of creating a new `ZephyrProtocol` object, and call the `addZephyrPacketEventListener()` method. The following code snippet shows an example implementation of this method, where the heart rate value is sent to the `BioHarnessHandler` in the form of a `LogicTupleEvent`, after checking that the sensor is attached to the strap.

```
@Override
public void Connected(ConnectedEvent<BTClient> eventArgs) {
  pckTypeRq.EnableGP(true);

  BTComms btComms = eventArgs.getSource().getComms();
  ZephyrProtocol protocol = new ZephyrProtocol(btComms, pckTypeRq);

  protocol.addZephyrPacketEventListener(new ZephyrPacketListener() {
```

```
    @Override
    public void ReceivedPacket(ZephyrPacketEvent zephyrPacketEvent) {
      ZephyrPacketArgs packetArgs = zephyrPacketEvent.getPacket();
      int packetID = packetArgs.getMsgID();
      byte[] dataArray = packetArgs.getBytes();

      switch (packetID) {
        case GENERAL_PACKET_ID:
          byte worn = gpPacketInfo.GetWornStatus(dataArray);
          if (worn == 1) { // Send values only when the device is
              attached to the strap
            Message msg = handler.obtainMessage();
            msg.arg1 = SensorHandler.SEND_MESSAGE;
            Bundle bundle = new Bundle();

            // Put the Heart Rate in the Bundle as a LogicTupleEvent
            int heartRateNum = gpPacketInfo.GetHeartRate(dataArray);
            String heartRate = String.valueOf(heartRateNum);
            LogicTupleEvent ev = new LogicTupleEvent(HEART_RATE,
                heartRate);
            bundle.putParcelable(MAGPIE_LTE_EVENT,ev);
            // Put the bundle in the message and send it back to the
                Handler
            msg.setData(bundle);
            handler.sendMessage(msg);
            break;
          }
        default:
          Log.e(TAG, "Packet type '" + packetID + "' from BioHarness
              sensor not processed");
      }
    }
  });
}
```

Appendix C

The Event Calculus

Contents

The Prolog Mind from the MAGPIE agent platform that was introduced in Chapter 4, and the monitoring rules that were introduced in Chapter 5 use the Event Calculus (EC) [89, 122] as the underlying formalism to deal with the temporal events produced in the agent environment. The EC is a formalism for representing actions and their effects, and therefore it is suitable to model expert systems representing the evolution in time of an entity by means of the production of events. As introduced in the thesis, in MAGPIE the EC reasoner is embedded inside an agent, and models the monitoring rules that can be defined by the medical doctors through the web application.

The EC is based on many-sorted first-order predicate calculus, known as domain independent axioms, which are represented as normal logic programs that are executable in Prolog. The underlying time model of the EC is linear. The EC manipulates fluents, where a fluent represents a property which can have different values over time. The term F=V denotes that the fluent F has value V, as a consequence of an action that took place at some earlier time-point and not terminated by another action in the meantime. Table C.1 summarizes the main EC predicates used. Predicates, function symbols and constants start with a lower-case letter, while variables starts with an upper-case letter.

The domain independent axioms of the EC are the following:

$$\text{holdsAt}(F = V, 0) \leftarrow \text{initiatially}(F = V). \tag{C.1}$$

Table C.1: Main Event Calculus predicates used

Predicate	**Meaning**
initially(F=V)	The value of fluent F is V at time 0
holdsAt(F=V,T)	The value of fluent F is V at time T
holdsFor(F=V,[Tmin,Tmax])	The value of fluent F is V between Tmin and Tmax
initiatesAt(F=V,T)	At time T the fluent F is initiated to have value V
terminatesAt(F=V,T)	At time T the fluent F is terminated from having value V
broken(F=V,[Tmin,Tmax])	The value of fluent F is either terminated at Tmax, or initiated to a different value than V between Tmin and Tmax
happensAt(E,T)	An event E takes place at time T updating the state of the fluents

$$\begin{aligned} &\text{holdsAt}(F = V, T) \leftarrow \\ &\qquad \text{initiatesAt}(F = V, T_s), T_s < T, \\ &\qquad \text{not broken}(F = V, [T_s, T]). \end{aligned} \tag{C.2}$$

Predicate (C.1) states that a fluent F holds value V at time 0, if it has been initially set to this value. For any other time T > 0, the predicate (C.2) states that the fluent holds at time T if it has been initiated to value V at some earlier time point Ts, and it has not been broken on the meanwhile.

$$\begin{aligned} &\text{broken}(F = V, [Tmin, Tmax]) \leftarrow \\ &\qquad \text{terminatesAt}(F = V, T), Tmin < T, Tmax > T. \end{aligned} \tag{C.3}$$

$$\begin{aligned} &\text{broken}(F = V_1, [Tmin, Tmax]) \leftarrow \\ &\qquad \text{initiatesAt}(F = V_2, T_i), V_1 \neq V_2, \\ &\qquad Tmin < Ti, Tmax > Ti. \end{aligned} \tag{C.4}$$

Predicates (C.3) and (C.4) specify the conditions that brake a fluent. Predicate (C.3) states that a fluent is broken between two time points Tmin and Tmax if within this interval it has been terminated to have value V. Alternatively, predicate (C.4) states that a fluent is broken within a time interval if it has been initiated to hold a different value.

$$\begin{aligned} &\text{holdsFor}(F = V, [Tmin, Tmax]) \leftarrow \\ &\qquad \text{initiatesAt}(F = V, Tmin), \\ &\qquad \text{terminiatesAt}(F = V, Tmax), \\ &\qquad \text{not broken}(F = V, [Tmin, Tmax]). \end{aligned} \tag{C.5}$$

$$\begin{aligned}\text{holdsFor}(F = V, [Tmin, infPlus]) \leftarrow \\ \text{initiatesAt}(F = V, Tmin), \\ \text{not broken}(F = V, [Tmin, infPlus]).\end{aligned} \tag{C.6}$$

$$\begin{aligned}\text{holdsFor}(F = V, [infMin, Tmax]) \leftarrow \\ \text{terminatesAt}(F = V, Tmax), \\ \text{not broken}(F = V, [infMin, Tmax]).\end{aligned} \tag{C.7}$$

Predicates (C.5), (C.6) and (C.7) deal with the validity intervals of fluents. In particular, predicate (C.5) specifies that a fluent F keeps value V for a time interval going from Tmin to Tmax if nothing happens in the middle that breaks such an interval. Predicates (C.6) and (C.7) behave in the same way, but deal with open intervals.

The domain dependent predicates in EC are typically expressed in terms of the initiatesAt/2 and terminatesAt/2 predicates. One example of a common rule for initiatesAt/2 is

$$\begin{aligned}\text{initatesAt}(F = V, T) \leftarrow \\ \text{happensAt}(Ev, T), \\ Conditions[T].\end{aligned} \tag{C.8}$$

The above definition states that a fluent is initiated to value V at time T if an event Ev happens at this time point, and some optional conditions depending on the domain are satisfied. In relation with MAGPIE, these events that must happen are physiological measurements from the patient.

C.1 Specific GDM Monitoring Rules in EC

According to the EC domain independent predicates introduced in the last section, the Gestational Diabetes Mellitus (GDM) alerts from Chapter 3 can be specified in terms of the initiatesAt/2 to build the following set of domain dependent rules for GDM.

- **Alert 1:** Hypoglycemia.
  - **Rule 1:** There are two consecutive days with glucose values less than 4 mmol/L in the same period of the day. This rule can be defined with the predicates (C.9) and (C.10).

$$\begin{aligned}\text{initiatesAt}(\text{glucose}(Period) = \text{low}, T) \leftarrow \\ \text{happensAt}(\text{measured\_glucose}(G), T), \\ G < 4.\end{aligned} \tag{C.9}$$

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{hypoglycemia}, Period) = \text{true}, T) \leftarrow \\
&\quad \text{one_day_time}(Tday), \\
&\quad \text{last_week}([Ts, T]), \\
&\quad \text{holdsFor}(\text{glucose}(Period) = \text{low}, [Ts, Te]), \\
&\quad Ts - Te \geq Tday.
\end{aligned} \tag{C.10}$$

- **Alert 2:** Severe hypoglycemia.

  - **Rule 2:** Two consecutive glucose values are less than 4 mmol/L within one hour.

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{severe_hypoglycemia}) = \text{true}, T) \leftarrow \\
&\quad \text{one_hour_time}(Thour), \\
&\quad \text{holdsAt}(\text{last_glucose_measurement}(G) = \text{time}(Tlast), T) \\
&\quad \text{happens_at}(\text{measured_glucose}(G), T), \\
&\quad Tdiff \text{ is } T - Tlast, \\
&\quad Tdiff \geq Thour, \\
&\quad G < 4.
\end{aligned} \tag{C.11}$$

- **Alert 3:** Postprandial hyperglycemia.

  - **Rule 3a:** Two times during the last four preceding days the glucose value is bigger or equal to 8 mmol/L in the periods after the meals.

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{postprandial_hyperglycemia}) = \text{active}, T) \leftarrow \\
&\quad \text{happensAt}(\text{glucose}(V1, P), T), \\
&\quad \text{last_four_days}(Time4Days, T), \\
&\quad (P = \text{after_breakfast}; \\
&\quad P = \text{after_lunch}; \\
&\quad P = \text{after_dinner}), \\
&\quad V1 \geq 8, \\
&\quad \text{count}((\text{happensAt}(\text{glucose}(V2, P), T2), \\
&\qquad T2 > Time4Days, \\
&\qquad T2 < T, V2 \geq 8), C), \\
&\quad C > 2.
\end{aligned} \tag{C.12}$$

  - **Rule 3b:** Three times during the last week the glucose value is bigger or equal to 7

mmol/L in the periods after the meals.

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{postprandial_hyperglycemia}) = \text{active}, T) \leftarrow \\
&\quad \text{happensAt}(\text{glucose}(V1, P), T), \\
&\quad \text{last_week}(\mathit{Time7Days}, T), \\
&\quad (P = \text{after_breakfast}; P = \text{after_lunch}; P = \text{after_dinner}), \\
&\quad V1 \geq 7, \\
&\quad \text{count}((\text{happensAt}(\text{glucose}(V2, P), T2), \\
&\qquad T2 > \mathit{Time7Days}, T2 < T, V2 \geq 7), C), \\
&\quad C > 3.
\end{aligned} \tag{C.13}$$

- **Alert 4:** Fasting hyperglycemia.

  - **Rule 4a:** Two times during the last four preceding days the glucose value is bigger or equal to 5.8 mmol/L in the same periods before the meals.

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{fasting_hyperglycemia}) = \text{active}, T) \leftarrow \\
&\quad \text{happensAt}(\text{glucose}(V1, P), T), \\
&\quad \text{last_four_days}(\mathit{Time4Days}, T), \\
&\quad P = \text{before_breakfast}, \\
&\quad V1 \geq 5.8, \\
&\quad \text{count}((\text{happensAt}(\text{glucose}(V2, P), T2), \\
&\qquad T2 > \mathit{Time4Days}, T2 < T, V2 \geq 5.8), C), \\
&\quad C > 2.
\end{aligned} \tag{C.14}$$

  - **Rule 4b:** Three times during the last week the glucose value is bigger or equal to 5.3 mmol/L in the same periods before the meals.

$$\begin{aligned}
&\text{initiatesAt}(\text{alert}(\text{fasting_hyperglycemia}) = \text{active}, T) \leftarrow \\
&\quad \text{happensAt}(\text{glucose}(V1, P), T), \\
&\quad \text{last_week}(\mathit{Time7Days}, T), \\
&\quad P = \text{before_breakfast}, \\
&\quad V1 \geq 5.3, \\
&\quad \text{count}((\text{happensAt}(\text{glucose}(V2, P), T2), \\
&\qquad T2 > \mathit{Time7Days}, T2 < T, V2 \geq 5.3), C), \\
&\quad C > 3.
\end{aligned} \tag{C.15}$$

Similarly to the more_or_equals_to/2 predicate introduced in Chapter 5, the count/2 predicate specifies the amount of times that the specified condition holds.

C.2 JSON Representation of Monitoring Rules

The monitoring rules discussed in this thesis follow a pattern, which makes them easy to represent in JSON. This format is used for their transmission over the network, and in the Tier 2 they are translated to its Prolog representation before being loaded into an agent's Prolog mind. The following code snippet shows an example for the rule in the Figure 5.5, whose Prolog representation is the predicate (5.1). This sequential rule defines the glucose pattern high $\rightarrow$ low $\rightarrow$ high. Notice that numerical codes are used to identify some elements, e.g. the one day temporal window is identified through the values of the keys timing and frequency. In this case, where a sequential rule is considered, the id key of the event objects identifies the order in which the events must be considered.

```
{
    "id": 0,
    "label": "sequential",
    "name": "alert(glucose_pattern)",
    "timing": 1,
    "frequency": 1,
    "events": [
        {
            "id": 0,
            "label": "glucose",
            "operands": [
                {
                    "type": 3,
                    "value": 8
                }
            ]
        },
        {
            "id": 1,
            "label": "glucose",
            "operands": [
                {
                    "type": 0,
                    "value": 3.8
                }
            ]
        },
        {
```

```
            "id": 2,
            "label": "glucose",
            "operands": [
                {
                    "type": 3,
                    "value": 8
                }
            ]
        }
    ]
}
```

Bibliography

[1] S. Abbate, M. Avvenuti, F. Bonatesta, G. Cola, P. Corsini, and A. Vecchio. A smartphone-based fall detection system. *Pervasive and Mobile Computing*, 8(6):883–899, 2012.

[2] A. Abdullah, Z. Zakaria, and N. Mohamad. Design and development of fuzzy expert system for diagnosis of hypertension. In *Proc. 2nd International Conference on Intelligent Systems, Modelling and Simulation (ISMS)*, pages 113–117, 2011.

[3] G. Acampora, D. Cook, P. Rashidi, and A. Vasilakos. A survey on ambient intelligence in healthcare. *Proceedings of the IEEE*, 101(12):2470–2494, 2013.

[4] J. Agüero, M. Rebollo, C. Carrascosa, and V. Julián. Does android dream with intelligent agents? In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50 of *Advances in Soft Computing*, pages 194–204. Springer Berlin Heidelberg, 2008.

[5] Amazon Web Services. `https://aws.amazon.com/`. Accessed: 01-10-2015.

[6] O. Amft and G. Tröster. On-body sensing solutions for automatic dietary monitoring. *IEEE Pervasive Computing*, 8(2):62 –70, 2009.

[7] B. Arnrich, O. Mayora, J. Bardram, and G. Tröster. Pervasive healthcare: Paving the way for a pervasive, user-centered and preventive healthcare model. *Methods of Information in Medicine*, 49(1):67–73, 2010.

[8] O. Arsene, I. Dumitrache, and I. Mihu. Expert system for medicine diagnosis using software agents. *Expert Systems with Applications*, 42(4):1825 – 1834, 2015.

[9] S. M. Arteaga, V. M. González, S. Kurniawan, and R. A. Benavides. Mobile games and design requirements to increase teenagers' physical activity. *Pervasive and Mobile Computing*, 8(6):900–908, 2012.

[10] J. C. Augusto. Temporal reasoning for decision support in medicine. *Artificial Intelligence in Medicine*, 33(1):1–24, 2005.

[11] S. Avancha, A. Baxi, and D. Kotz. Privacy in mobile technology for personal healthcare. *ACM Computing Surveys*, 45(1):3:1–3:54, 2012.

[12] O. Aziz, L. Atallah, B. P. L. Lo, E. Gray, T. Athanasiou, A. Darzi, and G.-Z. Yang. Ear-worn body sensor network device: an objective tool for functional postoperative home recovery monitoring. *Journal of the American Medical Informatics Association*, 18(2):156–159, 2011.

[13] J. E. Bardram. Pervasive healthcare as a scientific discipline. *Methods of Information in Medicine*, 47(3):178–185, 2008.

[14] J. E. Bardram and H. B. Christensen. Pervasive computing support for hospitals: An overview of the Activity-Based Computing project. *IEEE Pervasive Computing*, 6(1):44–51, 2007.

[15] J. E. Bardram, M. Frost, K. Szántó, and G. Marcu. The MONARCA self-assessment system: a persuasive personal monitoring system for bipolar patients. In *Proc. 2nd ACM SIGHIT International Health Informatics Symposium (IHI)*, pages 21–30, 2012.

[16] BaseX. The XML Database. `http://basex.org/`. Accessed: 01-10-2015.

[17] M. Baumgarten and M. D. Mulvenna. Cognitive sensor networks: Towards self-adapting ambient intelligence for pervasive healthcare. In *Proc. 5th International Conference of Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 366–369, 2011.

[18] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. John Wiley & Sons, 2007.

[19] A. Benharref, M. Serhani, and A.-R. Nujum. Closing the loop from continuous m-health monitoring to fuzzy logic-based optimized recommendations. In *Proc. 36th International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2698–2701, 2014.

[20] T. W. Bickmore, D. Mauer, and T. Brown. Context awareness in a handheld exercise agent. *Pervasive and Mobile Computing*, 5(3):226–235, 2009.

[21] BioHarness 3 Data Sheet. `http://www.zephyranywhere.com/media/pdf/BH_DS_P-BioHarness3-Data-Sheet_20120919_V02.pdf`. Accessed: 01-10-2015.

[22] M. Blount, M. Ebling, J. Eklund, A. James, C. McGregor, N. Percival, K. Smith, and D. Sow. Real-time analysis for intensive care: Development and deployment of the artemis analytic system. *IEEE Engineering in Medicine and Biology Magazine*, 29(2):110–118, 2010.

[23] J. Blum and E. Magill. Telecare service challenge: Conflict detection. In *Proc. 5th International Conference of Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 502–507, 2011.

[24] A. B. Bondi. Characteristics of scalability and their impact on performance. In *Proc. 2nd International Workshop on Software and Performance (WOSP)*, pages 195–203, 2000.

[25] K. Boone. *The CDA ™Book*. Springer, 2011.

[26] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[27] J. A. Botia, A. Villa, and J. Palma. Ambient Assisted Living system for in-home monitoring of healthy independent elders. *Expert Systems with Applications*, 39(9):8136–8148, 2012.

[28] S. Bromuri, M. Schumacher, K. Stathis, and J. Ruiz. Monitoring gestational diabetes mellitus with cognitive agents and agent environments. In *Proc. 2011 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 409–414, 2011.

[29] S. Bromuri and K. Stathis. Distributed agent environments in the ambient event calculus. In *Proc. 3rd ACM International Conference on Distributed Event-Based Systems (DEBS)*, 2009.

[30] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing*. Wiley, 2007.

[31] J. B. Buse, H. N. Ginsberg, G. L. Bakris, N. G. Clark, F. Costa, et al. Primary prevention of cardiovascular diseases in people with diabetes mellitus: a scientific statement from the American Heart Association and the American Diabetes Association. *Diabetes Care*, 30(1):162–172, 2007.

[32] M. Butler. Android: Changing the mobile landscape. *IEEE Pervasive Computing*, 10(1):4–7, 2011.

[33] I. Capel, M. Rigla, G. García-Sáez, A. Rodríguez-Herrero, B. Pons, D. Subías, F. García-García, M. Gallach, M. Aguilar, C. Pérez-Gandía, et al. Artificial pancreas using a personalized rule-based controller achieves overnight normoglycemia in patients with type 1 diabetes. *Diabetes Technology & Therapeutics*, 16(3):172–179, 2014.

[34] W. Carswell, J. Augusto, M. Mulvenna, J. Wallace, S. Martin, P. McCullagh, H. Zheng, H. Wang, K. McSorley, B. Taylor, and W. P. Jeffers. The NOCTURNAL Ambient Assisted Living system. In *Proc. 5th International Conference of Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 208–209, 2011.

[35] P. Cavalcante Aguilar, J. Boudy, D. Istrate, B. Dorizzi, and J. Moura Mota. A dynamic evidential network for fall detection. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1103–1113, 2014.

[36] L. Cervantes, Y.-S. Lee, H. Yang, S.-h. Ko, and J. Lee. Agent-based intelligent decision support for the home healthcare environment. In *Advances in Hybrid Information Technology*, volume 4413 of *Lecture Notes in Computer Science*, pages 414–424. Springer Berlin Heidelberg, 2007.

[37] Y.-S. Chang, C.-T. Fan, W.-T. Lo, W.-C. Hung, and S.-M. Yuan. Mobile cloud-based depression diagnosis using an ontology and a bayesian network. *Future Generation Computer Systems*, 43-44:87–98, 2015.

[38] L. Chen, X. Zhang, and C. Song. An automatic screening approach for obstructive sleep apnea diagnosis based on single-lead electrocardiogram. *IEEE Transactions on Automation Science and Engineering*, 12(1):106–115, 2015.

[39] Y. Chen, J. Zhang, and P. Pu. Exploring social accountability in pervasive fitness apps. In *Proc. 8th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, pages 221–226, 2014.

[40] L. Chittaro and M. Dojat. Using a general theory of time and change in patient monitoring: Experiment and evaluation. *Computers in Biology and Medicine*, 27(5):435–452, 1997.

[41] J. Choi, S. Yoo, H. Park, and J. Chun. MobileMed: A PDA-based mobile clinical information system. *IEEE Transactions on Information Technology in Biomedicine*, 10(3):627–635, 2006.

[42] C. Cobelli, E. Renard, and B. Kovatchev. Artificial pancreas: past, present, future. *Diabetes*, 60(11):2672–2682, 2011.

[43] C. Combi and Y. Shahar. Temporal reasoning and temporal data maintenance in medicine: Issues and challenges. *Computers in Biology and Medicine*, 27(5):353–368, 1997.

[44] Continua Health Alliance. `http://www.continuaalliance.org`. Accessed: 01-10-2015.

[45] D. J. Cook, J. C. Augusto, and V. R. Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298, 2009.

[46] J. M. Corchado, J. Bajo, Y. de Paz, and D. I. Tapia. Intelligent environment for monitoring alzheimer patients, agent technology for health care. *Decision Support Systems*, 44(2):382–396, 2008.

[47] E. Denti, A. Omicini, and A. Ricci. Multi-paradigm Java-Prolog integration in tuProlog. *Science of Computer Programming*, 57(2):217–250, 2005.

[48] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon. Gamification. using game-design elements in non-gaming contexts. In *Proc. 29th International Conference on Human Factors in Computing Systems, Extended Abstracts Volume (CHI)*, pages 2425–2428, 2011.

[49] R. H. Dolin, L. Alschuler, S. Boyer, C. Beebe, F. M. Behlen, P. V. Biron, and A. S. Shvo. HL7 clinical document architecture, release 2. *Journal of the American Medical Informatics Association*, 13(1):30–39, 2006.

[50] C. Doukas and I. Maglogiannis. Intelligent pervasive healthcare systems. In *Advanced Computational Intelligence Paradigms in Healthcare - 3*, volume 107 of *Studies in Computational Intelligence*, pages 95–115. Springer Berlin Heidelberg, 2008.

[51] M. ElHelw, J. Pansiot, D. McIlwraith, R. Ali, B. Lo, and L. Atallah. An integrated multisensing framework for pervasive healthcare monitoring. In *Proc. 3rd International Conference of Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 1–7, 2009.

[52] M. ElSayed, A. Alsebai, A. Salaheldin, N. El Gayar, and M. ElHelw. Ambient and wearable sensing for gait classification in pervasive healthcare environments. In *Proc. 12th IEEE International Conference on e-Health Networking Applications and Services (Healthcom)*, pages 240 –245, 2010.

[53] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[54] Europe 2020 - for a healthier EU. `http://ec.europa.eu/health/europe_2020_en.htm`. Accessed: 01-10-2015.

[55] A. Facchinetti, G. Sparacino, and C. Cobelli. An online self-tunable method to denoise cgm sensor data. *IEEE Transactions on Biomedical Engineering*, 57(3):634–641, 2010.

[56] A. Farooq, A. Jalal, and S. Kamal. Dense RGB-D map-based human tracking and activity recognition using skin joints features and self-organizing map. *KSII Transactions on Internet and Information Systems*, 9(5):1856–1869, 2015.

[57] B. J. Fogg. *Persuasive Technology: Using Computers to Change What We Think and Do*. Morgan Kaufmann Publishers, 2003.

[58] C. Frantz, M. Nowostawski, and M. K. Purvis. Augmenting Android with AOSE Principles for Enhanced Functionality Reuse in Mobile Applications. In *Advanced Agent Technology*, volume 7068 of *Lecture Notes in Computer Science*, pages 187–211. Springer Berlin Heidelberg, 2012.

[59] B. Franz, A. Schuler, and O. Krauss. Applying FHIR in an integrated health monitoring system. *European Journal for Biomedical Informatics*, 11(2):51–56, 2015.

[60] H. Garsden, J. Basilakis, B. Celler, K. Huynh, and N. Lovell. A home health monitoring system including intelligent reporting and alerts. In *Proc. 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEMBS)*, volume 2, pages 3151–3154, 2004.

[61] K. Gayathri, S. Elias, and B. Ravindran. Hierarchical activity recognition for dementia care using markov logic network. *Personal and Ubiquitous Computing*, 19(2):271–285, 2015.

[62] S. Ghose, J. Mitra, M. Karunanithi, and J. Dowling. Human activity recognition from smart-phone sensor data using a multi-class ensemble learning in home monitoring. In *Driving Reform: Digital Health is Everyone's Business*, volume 214 of *Studies in Health Technology and Informatics*, pages 62–67. IOS Press, 2015.

[63] J. Gurzoni, F. Cozman, M. Martins, and P. Santos. Logic-probabilistic model for event recognition in a robotic search and rescue scenario. In *Proc. 2014 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 1726–1731, 2014.

[64] S. Halko and J. A. Kientz. Personality and persuasive technology: an exploratory study on health-promoting mobile applications. In *Persuasive Technology*, volume 6137 of *Lecture Notes in Computer Science*, pages 150–161. Springer Berlin Heidelberg, 2010.

[65] T. Hansen, J. Bardram, and M. Soegaard. Moving out of the lab: Deploying pervasive technologies in a hospital. *IEEE Pervasive Computing*, 5(3):24–31, 2006.

[66] D. Hardt. The OAuth 2.0 Authorization Framework, 2012. RFC 6749. [Online]. Available: `http://tools.ietf.org/html/rfc6749`.

[67] Health Level Seven International. `http://www.hl7.org`. Accessed: 01-10-2015.

[68] L. Heinemann and L. Krinelke. Insulin infusion set: the achilles heel of continuous subcutaneous insulin infusion. *Journal of Diabetes Science and Technology*, 6(4):954–964, 2012.

[69] M. Hossain and G. Muhammad. Cloud-assisted speech and face recognition framework for health monitoring. *Mobile Networks and Applications*, 20(3):391–399, 2015.

[70] IEEE. IEEE Standard Computer Dictionary. A Compilation of IEEE Standard Computer Glossaries. *IEEE Std 610*, 1991.

[71] Integrating the Healthcare Enterprise. `http://www.ihe.net`. Accessed: 01-10-2015.

[72] International Classification of Diseases (ICD). `http://www.who.int/classifications/icd`. Accessed: 01-10-2015.

[73] ISO 13606-1. Health informatics - Electronic health record communication - Part 1: Reference model, 2008.

[74] ISO/HL7 21731. Health informatics - HL7 version 3 - Reference Information Model, 2006.

[75] ISO/HL7 27932. Data Exchange Standards - HL7 Clinical Document Architecture, Release 2, 2009.

[76] ISO/IEC/IEEE Health informatics - Personal health device communication - Part 20601: Application profile - Optimized exchange protocol, 2010.

[77] B. Jin, T. H. Thu, E. Baek, S. Sakong, J. Xiao, T. Mondal, and M. Deen. Walking-age analyzer for healthcare applications. *IEEE Journal of Biomedical and Health Informatics*, 18(3):1034–1042, 2014.

[78] Ö. Kafalı, S. Bromuri, M. Sindlar, T. van der Weide, E. Aguilar Pelaez, U. Schaechtle, B. Alves, D. Zufferey, E. Rodriguez-Villegas, M. I. Schumacher, and K. Stathis. COMMODITY$_{12}$: A smart e-health environment for diabetes management. *Journal of Ambient Intelligence and Smart Environments*, 5(5):479–502, 2013.

[79] Ö. Kafalı, M. Sindlar, T. van der Weide, and K. Stathis. ORC: an ontology reasoning component for diabetes. In *Proc. 2nd International Workshop on Artificial Intelligence and NetMedicine (NetMed)*, pages 71–80, 2013.

[80] V. Kalpana, S. T. Hamde, and L. M. Waghmare. ECG feature extraction using principal component analysis for studying the effect of diabetes. *Journal of Medical Engineering & Technology*, 37(2):116–126, 2013.

[81] U. Kampmann, L. R. Madsen, G. Skajaa, D. S. Iversen, N. Moeller, and P. Ovesen. Gestational diabetes: A clinical update. *World Journal of Diabetes*, 6(8):1065–1072, 2015.

[82] N. Katzouris, A. Artikis, and G. Paliouras. Event recognition for unobtrusive assisted living. In *Artificial Intelligence: Methods and Applications*, volume 8445 of *Lecture Notes in Computer Science*, pages 475–488. Springer International Publishing, 2014.

[83] K. Kawamoto, C. A. Houlihan, E. A. Balas, and D. F. Lobach. Improving clinical practice using clinical decision support systems: a systematic review of trials to identify features critical to success. *BMJ*, 330(7494):765, 2005.

[84] D. B. Keenan, J. J. Mastrototaro, G. Voskanyan, and G. M. Steil. Delays in minimally invasive continuous glucose monitoring devices: a review of current technology. *Journal of Diabetes Science and Technology*, 3(5):1207–1214, 2009.

[85] W. A. Khan, M. Hussain, A. M. Khattak, M. Afzal, B. Amin, and S. Lee. Integration of HL7 compliant smart home healthcare system and HMIS. In *Impact Analysis of Solutions for Chronic Disease Prevention and Management*, volume 7251 of *Lecture Notes in Computer Science*, pages 230–233. Springer Berlin Heidelberg, 2012.

[86] O. Kilic, A. Dogac, and M. Eichelberg. Providing interoperability of ehealth communities through peer-to-peer networks. *IEEE Transactions on Information Technology in Biomedicine*, 14(3):846–853, 2010.

[87] T. Kleinberger, M. Becker, E. Ras, A. Holzinger, and P. Müller. Ambient intelligence in assisted living: Enable elderly people to handle future interfaces. In *Universal Access in Human-Computer Interaction. Ambient Interaction*, volume 4555 of *Lecture Notes in Computer Science*, pages 103–112. Springer Berlin Heidelberg, 2007.

[88] V. Koutkias, I. Chouvarda, A. Triantafyllidis, A. Malousi, G. D. Giaglis, and N. Maglaveras. A personalized framework for medication treatment management in chronic care. *IEEE Transactions on Information Technology in Biomedicine*, 14(2):464–472, 2010.

[89] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[90] T. Kuroda, H. Sasaki, T. Suenaga, Y. Masuda, Y. Yasumuro, K. Hori, N. Ohboshi, T. Takemura, K. Chihara, and H. Yoshihara. Embedded ubiquitous services on hospital information systems. *IEEE Transactions on Information Technology in Biomedicine*, 16(6):1216–1223, 2012.

[91] J. Lebak, J. Yao, and S. Warren. HL7-compliant healthcare information system for home monitoring. In *Proc. 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEMBS)*, volume 2, pages 3338–3341, 2004.

[92] X. Liang, M. Barua, L. Chen, R. Lu, S. Shen, X. Li, and H. Luo. Enabling pervasive healthcare through continuous remote health monitoring. *IEEE Wireless Communications*, 19(6):10–18, 2012.

[93] H.-C. Lin, S.-Y. Chiang, K. Lee, and Y.-C. Kan. An activity recognition model using inertial sensor nodes in a wireless sensor network for frozen shoulder rehabilitation exercises. *Sensors*, 15(1):2181–2204, 2015.

[94] S.-C. Lin, Y.-L. Chiang, H.-C. Lin, J. Hsu, and J.-F. Wang. Design and implementation of a HL7-based physiological monitoring system for mobile consumer devices. In *Digest of Technical Papers International Conference on Consumer Electronics (ICCE)*, pages 1–2, 2009.

[95] Y. Liu and F. Li. PCA: A reference architecture for pervasive computing. In *Proc. 1st International Symposium on Pervasive Computing and Applications*, pages 99–103, 2006.

[96] M. Lluch-Ariet, A. Brugués, F. Vallverdú, and J. Pegueroles. Knowledge sharing in the health scenario. *Journal of Translational Medicine*, 12(Suppl 2):S8, 2014.

[97] Logical Observation Identifiers Names and Codes (LOINC). `http://www.loinc.org`. Accessed: 01-10-2015.

[98] A. Lounis, A. Hadjidj, A. Bouabdallah, and Y. Challal. Secure and scalable cloud-based architecture for e-health wireless sensor networks. In *Proc. 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7, 2012.

[99] S. L. Mabry, T. Schneringer, T. Etters, and N. Edwards. Intelligent agents for patient monitoring and diagnostics. In *Proc. of the 2003 ACM Symposium on Applied Computing (SAC)*, pages 257–262, 2003.

[100] S. Masoudi, N. Montazeri, M. Shamsollahi, D. Ge, A. Beuchee, P. Pladys, and A. Hernandez. Early detection of apnea-bradycardia episodes in preterm infants based on coupled hidden markov model. In *2013 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 243–248, 2013.

[101] A. Minutolo, G. Sannino, M. Esposito, and G. De Pietro. A rule-based mhealth system for cardiac monitoring. In *Proc. 2010 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pages 144–149, 2010.

[102] G. Mühl, L. Fiege, and P. Pietzuch. *Distributed Event-Based Systems*. Springer Science & Business Media, 2006.

[103] A. Mukherjee, A. Pal, and P. Misra. Data analytics in ubiquitous sensor-based health information systems. In *Proc. 6th International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST)*, pages 193–198, 2012.

[104] NEMA PS3 / ISO 12052. Health informatics - Digital imaging and communication in medicine (DICOM) including workflow and data management, 1993.

[105] N. Nirwal, N. Sardana, and A. Bhatt. Hopeful hearts: A mobile health care application. In *Proc. 7th International Conference on Contemporary Computing (IC3)*, pages 351–356, 2014.

[106] OECD work on health. `http://www.oecd.org/health/health-brochure.pdf`. Accessed: 01-10-2015.

[107] M. J. O'Grady, A. J. Retterath, D. B. Keenan, N. Kurtz, M. Cantwell, et al. The use of an automated, portable glucose control system for overnight glucose control in adolescents and young adults with type 1 diabetes. *Diabetes care*, 35(11):2182–2187, 2012.

[108] H. Peng, B. Hu, Q. Liu, Q. Dong, Q. Zhao, and P. Moore. User-centered depression prevention: An EEG approach to pervasive healthcare. In *Proc. 5th International Conference of Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 325–330, 2011.

[109] C. Pérez-Gandía, A. Facchinetti, G. Sparacino, C. Cobelli, E. J. Gómez, M. Rigla, A. De Leiva, and M. E. Hernando. Artificial neural network algorithm for online glucose prediction from continuous glucose monitoring. *Diabetes Technology & Therapeutics*, 12(1):81–88, 2010.

[110] G. Pioggia, G. Tartarisco, G. Valenza, G. Ricci, L. Volpi, G. Siciliano, and S. Bonfiglio. A pervasive activity management and rehabilitation support system for the elderly. In *Proc. 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 813–816, 2010.

[111] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. In *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent*

Systems, Artificial Societies, and Simulated Organizations, pages 149–174. Springer US, 2005.

[112] S. Prasad. Designing for scalability and trustworthiness in mHealth systems. In *Distributed Computing and Internet Technology*, volume 8956 of *Lecture Notes in Computer Science*, pages 114–133. Springer International Publishing, 2015.

[113] S. Prescher, A. Bourke, F. Koehler, A. Martins, H. Sereno Ferreira, T. Boldt Sousa, R. Castro, A. Santos, M. Torrent, S. Gomis, M. Hospedales, and J. Nelson. Ubiquitous ambient assisted living solution to promote safer independent living in older adults suffering from co-morbidity. In *Proc. 34th International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5118–5121, 2012.

[114] D. Ravish, N. Shenoy, K. Shanthi, and S. Nisargh. Heart function monitoring, prediction and prevention of heart attacks: Using artificial neural networks. In *Proc. 1st International Conference on Contemporary Computing and Informatics (IC3I)*, pages 1–6, 2014.

[115] V. Rialle, J.-B. Lamy, N. Noury, and L. Bajolle. Telemonitoring of patients at home: a software agent approach. *Computer Methods and Programs in Biomedicine*, 72(3):257 – 268, 2003.

[116] A. Ricci, M. Viroli, and A. Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In *Environments for Multi-Agent Systems III*, volume 4389 of *Lecture Notes in Computer Science*, pages 67–86. Springer Berlin Heidelberg, 2007.

[117] C. Roe and S. Gonik. Server-side design principles for scalable internet systems. *IEEE Software*, 19(2):34–41, 2002.

[118] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[119] L. Sacchi, G. Lanzola, N. Viani, and S. Quaglini. Personalization and patient involvement in decision support systems: Current trends. *IMIA Yearbook*, 10(1):106–118, 2015.

[120] A. Santi, M. Guidi, and A. Ricci. JaCa-Android: An agent-based platform for building smart mobile applications. In *Proc. 3rd International Workshop on Languages, Methodologies, and Development Tools for Multi-Agent Systems (LADS)*, pages 95–114, 2010.

[121] B. D. Sekar and M. Dong. Function formula oriented construction of bayesian inference nets for diagnosis of cardiovascular disease. *BioMed Research International*, 2014:376378, 2014.

[122] M. Shanahan. The event calculus explained. In *Artificial Intelligence Today*, volume 1600 of *Lecture Notes in Computer Science*, pages 409–430. Springer Berlin Heidelberg, 1999.

[123] J. Shaw, R. Sicree, and P. Zimmet. Global estimates of the prevalence of diabetes for 2010 and 2030. *Diabetes Research and Clinical Practice*, 87(1):4–14, 2010.

[124] SNOMED CT. `http://www.ihtsdo.org/snomed-ct`. Accessed: 01-10-2015.

[125] K. Stathis, A. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In *Proc. 4th International Symposium "From Agent Theory to Agent Implementation" (AT2AI-4)*, pages 523–528, 2004.

[126] C.-J. Su and C. W. Peng. Multi-agent ontology-based web 2.0 platform for medical rehabilitation. *Expert Systems with Applications*, 39(12):10311–10323, 2012.

[127] H. Sun, V. De Florio, N. Gui, and C. Blondia. Promises and challenges of ambient assisted living systems. In *Proc. 6th International Conference on Information Technology: New Generations (ITNG)*, pages 1201–1207, 2007.

[128] C. Sunil Kumar, C. Guru Rao, and A. Govardhan. A framework for interoperable healthcare information systems. In *2010 International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, pages 604–608, 2010.

[129] M. Swan. Emerging patient-driven health care models: An examination of health social networks, consumer personalized medicine and quantified self-tracking. *International Journal of Environmental Research and Public Health*, 6(2):492–525, 2009.

[130] A. Tablado, A. Illarramendi, M. I. Bagüés, J. Bermúdez, and A. Goñi. Aingeru: an innovating system for tele assistance of elderly people. In *Proc. 1st International Workshop on Tele-Care and Collaborative Virtual Communities in Elderly Care (TELECARE)*, pages 27–36, 2004.

[131] M. Tauschmann and R. Hovorka. Insulin pump therapy in youth with type 1 diabetes: toward closed-loop systems. *Expert Opinion on Drug Delivery*, 11(6):943–955, 2014.

[132] A. Toninelli, R. Montanari, and A. Corradi. Enabling secure service discovery in mobile healthcare enterprise networks. *IEEE Wireless Communications, IEEE*, 16(3):24–32, 2009.

[133] F. Touati and R. Tabish. U-healthcare system: State-of-the-art review and challenges. *Journal of Medical Systems*, 37(3):9949, 2013.

[134] A. Triantafyllidis, C. Velardo, T. Chantler, S. A. Shah, C. Paton, R. Khorshidi, L. Tarassenko, and K. Rahimi. A personalised mobile-based home monitoring system for heart failure: The SUPPORT-HF study. *International Journal of Medical Informatics*, 84(10):743–753, 2015.

[135] M. Ughetti, T. Trucco, and D. Gotta. Development of agent-based, peer-to-peer mobile applications on Android with JADE. In *Proc. 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, pages 287–294, 2008.

[136] U. Varshney. Pervasive healthcare. *IEEE Computer*, 36(12):138–140, 2003.

[137] U. Varshney. Pervasive healthcare and wireless health monitoring. *Mobile Networks and Applications*, 12(2-3):113–127, 2007.

[138] M. Velikova, J. T. van Scheltinga, P. J. Lucas, and M. Spaanderman. Exploiting causal functional relationships in bayesian network modelling for personalised healthcare. *International Journal of Approximate Reasoning*, 55(1, Part 1):59–73, 2014.

[139] M. Vidal, J. Turner, A. Bulling, and H. Gellersen. Wearable eye tracking for mental health monitoring. *Computer Communications*, 35(11):1306–1311, 2012.

[140] M. Wallymahmed. Encouraging people with diabetes to get the most from blood glucose monitoring: Observing and acting upon blood glucose patterns. *Journal of Diabetes Nursing*, 17(1):6–13, 2013.

[141] D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.

[142] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2009.

[143] L. Xu, D. Guo, F. E. H. Tay, and S. Xing. A wearable vital signs monitoring system for pervasive healthcare. In *Proc. 2010 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (STUDENT)*, pages 86 –89, 2010.

[144] M. Yu, A. Rhuma, S. Naqvi, L. Wang, and J. Chambers. A posture recognition-based fall detection system for monitoring an elderly person in a smart home environment. *IEEE Transactions on Information Technology in Biomedicine*, 16(6):1274–1286, 2012.

[145] B. Yuan and J. Herbert. Transparency issues in a hybrid reasoning architecture for assistive healthcare. *AASRI Procedia*, 4:268–274, 2013.

[146] K. Zarkogianni, K. Mitsis, M.-T. Arredondo, G. Fico, A. Fioravanti, and K. Nikita. Neuro-fuzzy based glucose prediction model for patients with type 1 diabetes mellitus. In *Proc. 2nd IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 252–255, 2014.

[147] Zephyr Website. `http://zephyranywhere.com/products/bioharness-3/`. Accessed: 1-10-2015.

[148] P. Zhang, X. Zhang, J. Brown, D. Vistisen, R. Sicree, J. Shaw, and G. Nichols. Global healthcare expenditure on diabetes for 2010 and 2030. *Diabetes Research and Clinical Practice*, 87(3):293 – 301, 2010.

[149] L. Zhou and G. Hripcsak. Temporal reasoning with medical data - a review with emphasis on medical natural language processing. *Journal of Biomedical Informatics*, 40(2):183–202, 2007.